1065

A165949

RD&E CENTER

Technical Report

No. 13132

Theoretical Development and Application of Discrete Time Quantized Data Controllers

(Phase I)

Contract Number DAAE07-84-C-R055

January 1986

By Dr. R. P. Judd and P. L. McIntosh
School of Engineering & Computer Science
Oakland University
Rochester, MI 48063

20020723210

AN-41579

U.S. ARMY TANK-AUTOMOTIVE COMMAND
RESEARCH, DEVELOPMENT & ENGINEERING CENTER
Warren, Michigan 48397-5000

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for Public Release: Distribution is Unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) 13132	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Oakland University School of Engr & Cmptr Science; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Rochester, MI 48063	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Analytical & Physical Simulation Branch; 8b. OFFICE SYMBOL (If applicable) AMSTA-RYA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) USATACOM Bldg 215 Warren, MI 48397-5000	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. -; PROJECT NO.; TASK NO.; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Theoretical Development and Application of Descrete Time Quantized Data Controllers (Phase I)

12. PERSONAL AUTHOR(S)
Dr. R. PM Judd and P. L. McIntosh

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 6/84 TO 6/85	86 Jan	186

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Controllers; Discrete Time Quantized Data Controllers; Table Look-up Technique; DTQD Controllers; M60 Elevation Controller; Grid Embedding

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
A new approach to feedback control based on a table look-up technique is developed. A grid embedding technique is used which maintains high accuracy with minimal table size. This report describes the use of the new control scheme as a regulator. A circuit which implements the control scheme is developed. This circuit is simpler, cheaper, faster, and more reliable than circuits developed for comparable controllers using traditional control theory.

This report is divided into four major sections. The first section derives the theoretical foundation for the new control techniques. Next, the operation of a computer program which aids in the design of these controllers is decribed. The last two sections develop a controller for the gun elevation system of an M60 tank. Finally, a complete listing and documentation of the computer program used in the design of the controller are included in the appendices.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL James L. Overholt	22b. TELEPHONE (Include Area Code) (313)574-5378; 22c. OFFICE SYMBOL AMSTA-RYA

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

THIS PAGE LEFT BLANK INTENTIONALLY

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

THIS PAGE LEFT BLANK INTENTIONALLY

1.0. INTRODUCTION

This report summarizes the work done by Oakland University on Army contract DAAE07-84-Q-R083. Oakland University explored a new method of state feedback control designed to regulate the output of continuous systems. The new approach, based on a table lookup technique, results in a controller which is faster, less complicated, less expensive, and more reliable than present military controllers. Since the new method models both the discretion of time and the quantization of state, it is referred to as Discrete Time Quantized Data (DTQD) system theory. This theory is still being developed. The main objective of the first stage of the research effort is to develop the theory so controllers can be designed to regulate systems. This has been done.

The discussion portion of the report is divided into four major sections. The first section develops the theoretical foundation needed to design controllers based on the new technique. Next, the software package which has been developed to aid in the design of these controllers is outlined. The last two sections apply the theory to a typical military application. The first of these two sections derives a mathematical model of the gun elevation system of the M60 tank. The next section uses the model to develop and simulate a controller for the elevation system. The response of the controller is examined. As might be expected, the controller responded quite favorably as a regulator. However, when sinusoidal disturbances are applied to the tank hull, the controller did not damp the disturbances as well as current technology. It was not designed to. Research is continuing in the area to improve the disturbance canceling characteristics of the controller and provide tracking abilities. The software package used to design the controllers is completely documented in an appendix. Finally, the complete source listing of the software package is provided in another appendix.

2.0. OBJECTIVES

The objectives for this research are as follows:

1) Develop and refine a new application of control theory based on look-up table techniques and the effects of state quantization in digitally implemented control.
2) Develop DTQD analogs of controllability and observability of systems.
3) Determine the improvements in system response, ease of implementation, and system reliability given this methodology.

3.0. CONCLUSIONS

This research is very promising. The theory needed to design a DTQD controller has been completed. A computer program to aid in the design of these controllers has been developed. The theory has been applied to a military application and the system simulated. The results showed that the theory worked quite well in regulating the system, but when disturbances were added the response became noisy. This was not entirely surprising since the theory behind the design of the controller was not developed to reject disturbances. However, a slight modification of the scaling algorithm should reduce the magnitude of the noise. This idea is suggested within the report and should be further developed.

The analogs to controllability and observability for DTQD systems are not addressed in this report. This area is currently being explored and results will be forthcoming.

The regulation of systems without system or measurement noise using DTQD controllers seems to be comparable to traditional control methods. A complete discussion on the implementation of the technology into digital hardware is in the body of this report. Since this circuitry is extremely simple, the resulting DTQD controller will be more reliable, less complicated, faster, and cheaper than the controllers using traditional technology.

4.0. RECOMMENDATIONS

Because of the success of initial research into DTQD controllers, a follow on project should be conducted. The research should focus on noise rejection, tracking abilities, output driven controllers (instead of the current state feedback structure), and application of the theory to large scale systems. After the research is completed, a particular application should be designated by the military to implement a DTQD controller, and an actual controller should be built and tested.

5.0. DISCUSSION

5.1. <u>DTQD Theory</u>

5.1.1. Introduction.

Traditionally there have been two approaches to the digital control of systems. The first method finds the discrete time model of the plant and then determines a controller which will regulate the output. Both classical (using Z transforms) and optimal control techniques have been well developed for this approach. In the second method, usually reserved for converting existing continuous controllers to digital controllers, the designer tries to emulate a continuous controller by digital circuits. It is not clear that either of these methods is the best strategy for using digital electronics to control a plant.

An alternative approach [1-3] to controlling digital systems is presented here. The prime consideration in deriving the new control structure is to develop a circuit which naturally incorporates the unique features of digital electronics. The new approach creates a "digital model" for the system. This new model describes the relation between the digital inputs and outputs of the system. That is, the effects of the data converters are an integral part of the modeling process, see Fig. 5-1. Once this is done, the controller for the system could be naturally implemented by a digital circuit. The controller is essentially a table look-up technique easily constructed from digital circuit elements.

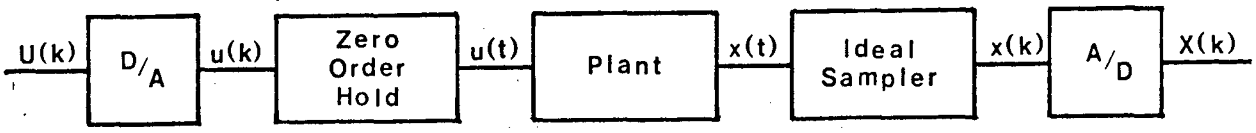


Figure 5-1. System Model

Other authors [4-6] have explored developing a digital model for continous systems. They have given up for two main reasons. First, there is the famous problem of the "curse of dimensionality." That is, the size of the control table will increase exponentially with the number of states. To accurately control even a second order system by this method requires huge tables. However, this problem can be minimized by using the grid embedding technique proposed in this report. The second problem with digital models is that in general its output will diverge from the actual system output. However, with proper selection of the quantization levels and sampling interval, the rate of divergence can be controlled. Since the primary purpose of the current research is to develop a feedback controller for the digital system, then a model which adequately describes the system for only one sample increment will be sufficient to develop a good responsive controller.

This portion of the report is divided into several sections. Section 5.1.2 develops the digital model for a continuous linear plant. It also shows that a digraph can be used to represent the digital model. Once this is done, the classic graph theory algorithms can be used to determine the control law. This is examined in Section 5.1.3. Section 5.1.4 discusses the dimensionality problem and suggests a solution. Section 5.1.5 illustrates the electronics needed to implement the controller. Finally, the last section suggests how this method might be extended to nonlinear systems. An example is also presented.

5.1.2. Quantization Theory.

Consider the system illustrated in Fig. 5-1. We wish to find the relation between the digital signals U(k) and X(k). First, assume that the plant is a linear system, that is

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{5-1}$$

where

$x \varepsilon R^n$

$u \varepsilon R^p$

Modeling the effect of time discretation is quite easy. Using standard linear system theory the relation between x(k) and u(k) is represented by

$$x(k+1) = \Phi x(k) + Du(k) \qquad (5\text{-}2)$$

where

$$\Phi = e^{AT}$$

$$D = \int_0^T e^{A(T-s)} B \, ds$$

T = the sampling period

Now the data converters must be included into the model. To do this a convention must be established to represent digital signals. Suppose there are j bits in a digital signal, then there are 2^j unique pieces of information that can be represented by the digital signal. We shall use the set of integers $[(-2^{j-1}),\ldots -1,0,1,\ldots(2^{j-1}-1)]$ to denote each piece of information.

Now examine the D/A converter. Its job is to convert p digital signals to p discrete signals. This can be easily done by multiplying each element of the U(K) vector by an appropriate scaling factor.

$$u(k) = \begin{bmatrix} \gamma_1 & & & & 0 \\ & \gamma_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ 0 & & & & \gamma_p \end{bmatrix} U(k) \qquad (5\text{-}3)$$

$$= \Gamma \quad U(k)$$

Recall that the digital input is modeled by a set of integers, that is U(k) is a vector of integers. Therefore, all the D/A converter is doing is mapping the integers U(k) to a vector of real numbers u(k) according to the scaling law represented in (5-3). Combining (5-2) and (5-3) we obtain

$$x(k+1) = \Phi x(k) + D\Gamma U(k) \qquad (5\text{-}4)$$

The A/D converter does the reverse job - it must convert the real numbers in the state vector to integers. For most converters, this process can be represented by

$$X(k) = \text{floor}\ (x(k)/\delta) \qquad (5\text{-}5)$$

or if X is a vector

$$X_i(k) = \text{floor}\ (x_i(k)/\delta_i)$$

$$i = 1,2,\ldots n \qquad (5\text{-}6)$$

Many converters may also include an offset ρ, i.e. $X = \text{floor}\ ((x+\rho)/\delta$. For the purposes of this paper ρ is assumed to be 0. This is done for clarity only. It does not alter any of the results. Let Δ designate this quantizing operation, that is $X(k) = \Delta x(k)$, then

$$X(k+1) = \Delta x(k+1) = \Delta[\Phi x(k) + D\Gamma U(k)] \qquad (5\text{-}7)$$

Unfortunately, Δ is <u>not</u> a linear operator, therefore the right side of (1-7) cannot be reduced. In fact the following argument will show that X(k+1), in general, <u>cannot</u> be represented as a function of X(k) and U(k).

Consider a system with only two states, then the data quantization process can be thought of as overlaying a lattice on top of the state space. Every state x(k) which resides in a single cell of the resulting grid belongs to the same quantized (or digital) state. The quantized state X(k) is then the n-dimensional integer vector representing the address of the cell. For example, examine the situation in Fig. 5-2. Here all of the states in the shaded portion of the state space are assigned the same quantized state $X(k) = (2,3)^t$. The problem comes after the system makes its transition to x(k+1). Suppose we trace each state in the shaded cell for one transition under a given input U(k). If x(k) was in the shaded cell at time k, then at time k+1 it must be in the parallelogram abcd. Unfortunately, this parallelogram overlaps four distinct cells. So, the X(k+1) <u>cannot</u> be deduced from knowing only U(k) and X(k). In other words, we do not have a state-determined system. However, knowledge of X(k) does reveal quite a bit about what X(k+1) can be. For example, in Fig. 5-2, if $X(k) = (1,3)^t$ then X(k+1) must be either $(3,1)^t$, $(3,2)^t$, $(4,1)^t$ or $(4,2)^t$. Now if the quantization is small enough, then transition can be modeled fairly accurately by picking any one of the four cells as the actual transition. It can be shown [3] that the number of cells that are overlapped, after a cell makes a transition, can be limited with proper selection of the sampling interval T and the quantization step size δ_i. Thus, we can develop a digital model of the system which, although not exact, will never be more than one cell in error in predicting the state for the next transition.

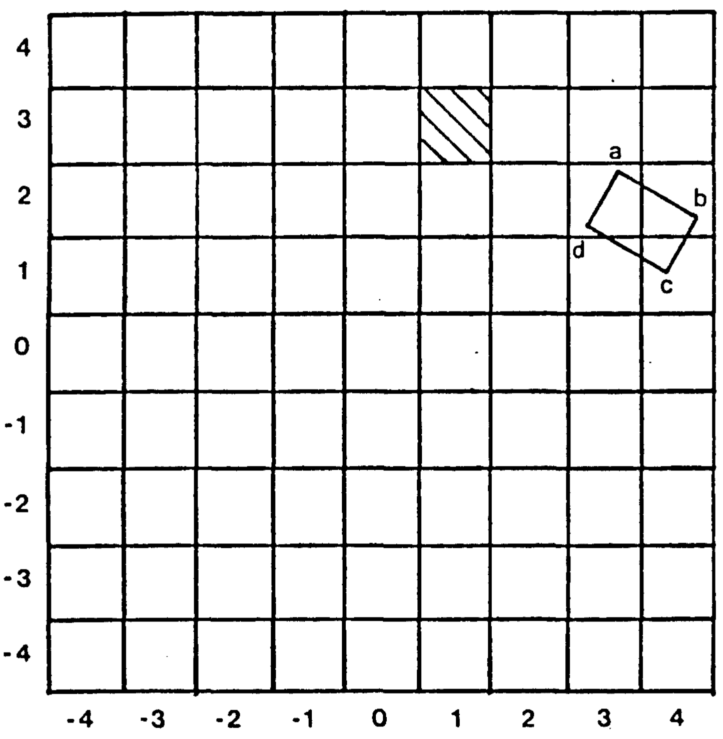


Figure 5-2. Quanization of a Two Dimensional Space

To formalize the mathematical definition of the model, we will trace only a single point in the cell, namely the center. So, for modeling purposes only, we will let

$$X(k+1) = \Delta(\Phi y + D\Gamma U(k)) \tag{5-8}$$

where

y = the center of the cell X(k).

Using this we can develop a state determined digital model ψ for the system.

$$X(k+1) = \psi[X(k), U(k)] \tag{5-9}$$

As was mentioned before, this model is not exact but with appropriate selection of T and δ_i's will predict X(k+1) to within one cell. This reseach is primarily concerned with developing a state feedback controller for the system. Since the controller can sense the state at every time interval, developing the control law based on this approximate model should yield satisfactory results. In fact, this model provided good results in the systems we have applied it to.

5.1.3. Control Law.

Consider a graph S whose vertices (nodes) are used to represent each cell in the discretation lattice. The edges in S then form the set of all possible transitions between the cells. For example, look at the digraph in Fig. 5-3. This graph represents a simple system. If the state is $(0,1)^t$ at time k and an input of 0 is applied, then the state will be $(0,0)^t$ at time k+1.

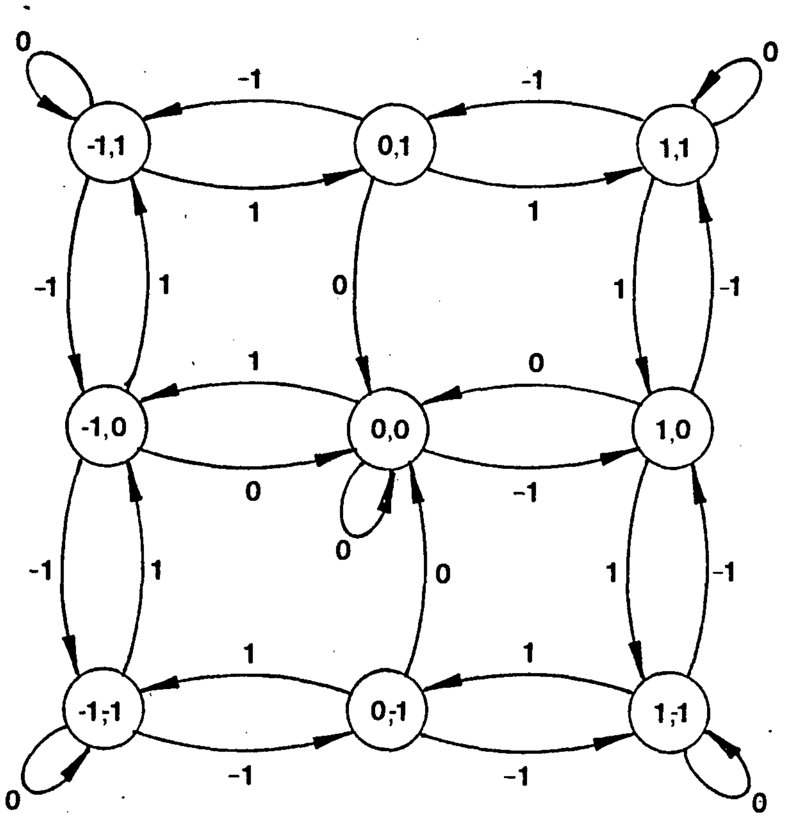


Figure 5-3. Digraph Representation

We now examine the possibility of controlling the system. Using the example presented in Fig. 5-3, we see that a good control law might be

$$U(k) = \begin{cases} 1; \text{ if } X(k) = (1,1),(-1,1),(1,-1), \\ \qquad\qquad\qquad (-1,-1) \\ 0; \text{ otherwise} \end{cases} \tag{5-10}$$

Using this law, the system reaches and remains in state (0,0) in minimum time.

To formalize an algorithm to determine the control law, consider the following cost functional

$$J = \sum_{k=0}^{N} C_U(U(k)) + C_X(X(k)) \tag{5-11}$$

where C_U and C_X are two non-negative functions of U and X respectively. The optimal control law of the sytem U(k) = F (X(k)) is then defined as the control U(k) which must be applied at each time k = 0,1,...N so that J is minimized. This formulation resembles traditional optimal control. This was done intentionally because we can use the same interpretations of C_U and C_X to come up with suitable control algorithms. For example, if

$$\begin{aligned} C_U(U(k)) &= 1 \\ C_X(X(k)) &= 0 \end{aligned} \tag{5-12}$$

then we have a minimum time system. If

$$\begin{aligned} C_U(U(k)) &= \text{abs}(U(k)) \\ C_X(X(k)) &= 0 \end{aligned} \tag{5-13}$$

then we will have a minimum energy system. Finally, even a linear quadratic regulator problem can be formulated by

$$\begin{aligned} C_U(U(k)) &= U^t(k)\ R\ U(k) \\ C_X(X(k)) &= X^t(k)\ Q\ X(k) \end{aligned} \tag{5-14}$$

where R is a positive definite matrix and Q is a positive semidefinite matrix.

The choice for representing the digital model now becomes apparent. The optimal control law formulation presented by (5-11) is exactly the same problem graph theorists refer to as the "optimal spanning tree" problem, where C_U is used to weight each of the edges and C_X weights all of the vertices in the graph. Already, there are well-defined algorithms to solve this problem [7-8]. We can use these algorithms directly to find F(X(k)).

The calculation of F(X(k)) can all be done off line. Once F(X(k)) is known, it can be stored in a PROM. The optimal control can then be found by addressing the PROM with the measured state X(k). This leads to an extremely simple implementation of the control law.

5.1.4. Dimensionality.

This approach suffers from the "curse of dimensionality." For example, suppose we have a system with three states, where each state is quantized into $1024 = 2^{10}$ levels. Then the capacity of the PROM needed to store the control algorithm is $(2^{10})^3$ or roughly one billion words. This is clearly too much memory to expend for the control of a relatively simple system.

This difficulty can be overcome by a <u>grid</u> <u>embedding</u> technique. Initially the state space is divided into a rather course grid. When the state is far from the origin, these large divisions are adequate. As the state is driven toward the origin, however, greater accuracy is required. This is achieved by mapping a small central region near the origin of the state space into the structure of the original discrete configuration. The process is continued until the desired accuracy is obtained.

This situation is depicted in Fig. 5-4. As the state moves into the center sixteen cells, the quantization level is cut in half, which results in the center 16 cells being mapped into the 64 cell structure of the original system. Since the embedding process will not occur until the state is within the specified central region, then the state must be somewhere in the 64 smaller cells created after the embedding process. So, at any time the controller needs to examine only 64 cells to derived its control strategy; however, after each embedding the size of the cells are cut by one-fourth. Thus the controller can achieve high precision with a relatively small table.

The embedding process will provide sufficient precision, even with relatively few cells in the state space. However, when the system is to be represented by just a few cells, the non-linearities of the quatization become significant. A way of modeling the non-linearities must be developed. The digital model proposed in this paper describes these non-linearities.

5.1.5 Implementation.

Suppose we wish to implement a controller for a second order system in which each state is divided into 16 divisions, i.e., there are a total of

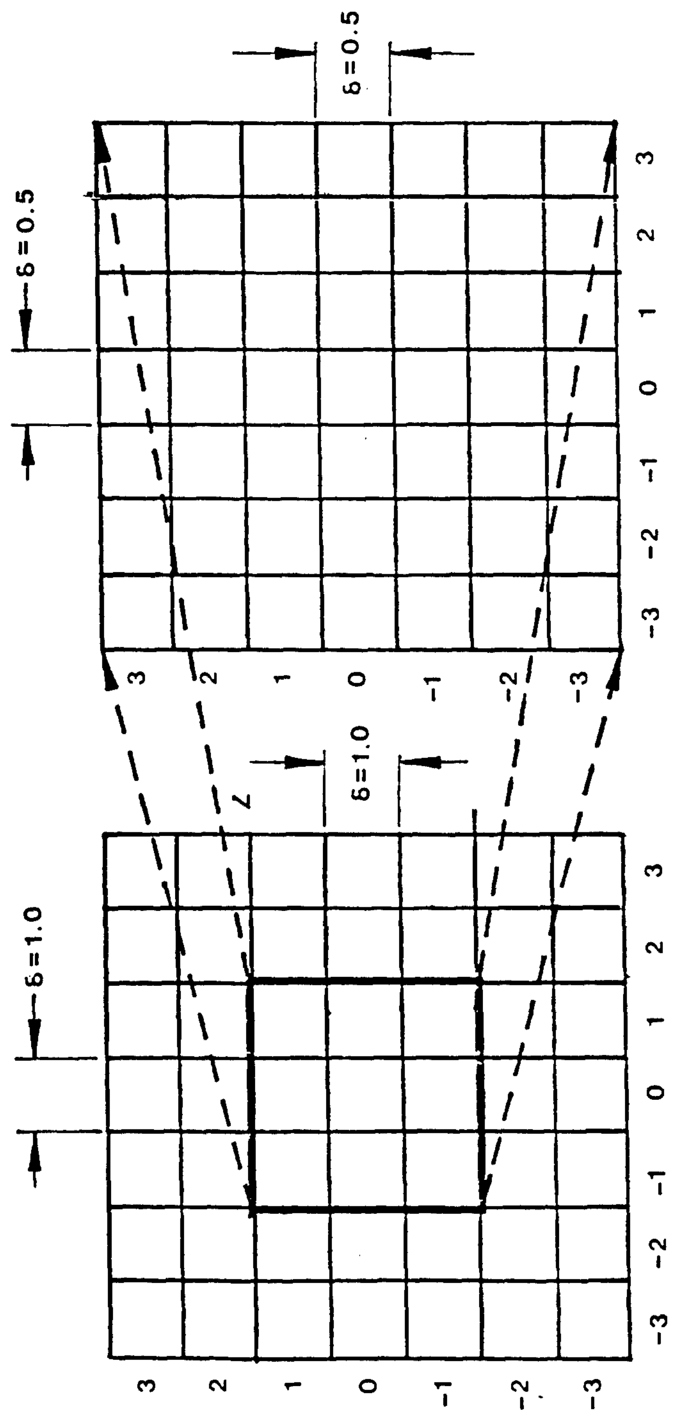


Figure 5-4. The Grid Embedding Technique

256 cells representing the entire state space. Embedding will take place whenever the state is within the center 16 cells. Each time the embedding process takes place, assume the quatitization levels are halved. Under these assumptions the embedding process can be easily implemented in hardware with shift registers.

To see this, examine Fig. 5-5. Both states are sampled and quantized to 10 bits of precision. The shift registers are set to pass the four most significant bits to the PROM which stores the control law. As the state is driven towards the center of the state space, the most significant bits of X_1 and X_2 are zeroed out. (If the A/D converters output numbers in two's compliment format, then the most significant bits become either zeros, for positive numbers, or ones, for negative numbers. In either case the circuit could tell when the system is approaching the center cells by exclusive-oring the most significant bits of X_1 and X_2.) When the two most significant bits of both X_1 and X_2 are all zeros or ones, then the Shifter Control Unit will instruct each register to shift right one bit. That is, bits b_1 - b_4 of X_1 and X_2 are used to drive the PROM instead of bits b_0 - b_3. This is equivalent to scaling each state quatization level by one half. The shift register to the right of the PROM will appropriately scale the input to the system. So, the grid embedding process can be easily implemented using a simple shifting technique.

It can be shown, [3], that the same control PROM can be used before and after embedding. Thus, a PROM which contains 256 words is sufficient for this controller. Also, the shifter control unit should be designed to continously monitor all the bits coming out of the A/D converters. This is needed for the following situation. Suppose a disturbance is encountered which will drive the state outside the bounds of an embedded grid. If the controller can detect this situation, it can expand the grid (by shifting left) to an appropriate size to capture the disturbance, and then procede as normal.

5.1.6. Non-Linear Systems.

In the development of this theory we explicitly assume that the system to be control is linear. However, this is not necessary. We can, with only a slight modification, use the theory on non-linear systems. The states still can be discretized and the digital model found by tracing the transition of the center of each cell. Furthermore, the optimal spanning tree algorithm makes no assumption about the graph it is being applied to. The only change which is necessary for non-linear systems exist in the grid embedding technique. For the non-linear systems, a new control law (PROM) may have to be switched in each time the embedding process is done.

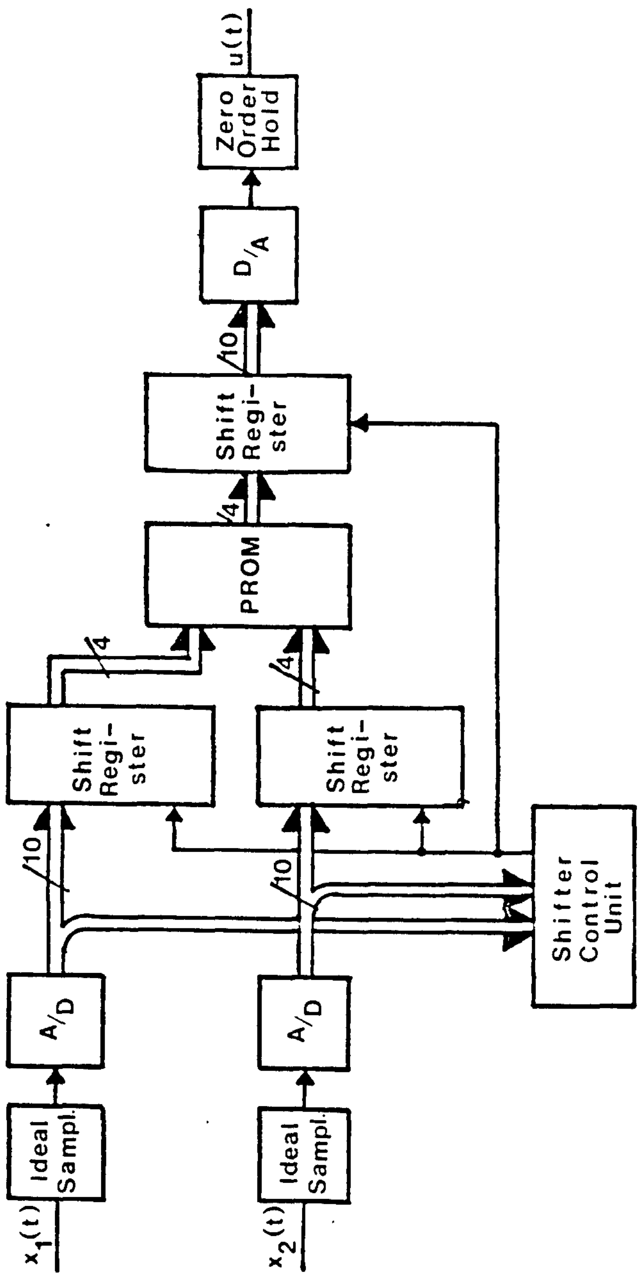


Figure 5-5. Hardware Implementation

5.1.7. Example.

This control algorithm has been applied to the following system, in which the state $\underline{x}(t)$ is to be regulated to the zero state.

$$\dot{x}(t) = \begin{bmatrix} 0.0 & 1.0 \\ 0.0 & -10.9 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 10.9 \end{bmatrix} u(t) \qquad (5\text{-}15)$$

We choose:

$$\begin{aligned} T &= 0.1 \quad \text{sec} \\ \delta_1 &= 2\pi/32 \quad \text{rad} \\ \delta_2 &= 1.5 \quad \text{rad/sec} \\ \gamma &= 1.25 \quad \text{volts} \end{aligned} \qquad (5\text{-}16)$$

and impose the following bounds on the states and inputs:

$$\begin{aligned} |x_1| &\leq \pi \quad \text{rad} \\ |x_2| &\leq 12 \quad \text{rad/sec} \\ |u| &\leq 10 \quad \text{volts} \end{aligned} \qquad (5\text{-}17)$$

the digital model for this system was derived, and using the "optimal spanning tree" algorithm with the following weights:

$$\begin{aligned} C_U &= U(k)^2 \\ C_X &= 2(x_1(k)^2 + x_2(k)^2) \end{aligned} \qquad (5\text{-}18)$$

the control law was developed. The embedding process was designed to proceed whenever the system is in any of the 32 center cells.

Figure 6 illustrates the simulated runs of this digital control strategy. For comparision, the trajectory for an optimal linear regulator using continuous state feedback is included. The performance index used for the continuous controller is given by

$$J = \int_0^{\infty} (u(t)^2 + 2x_1(t)^2 + 2x_2(t)^2)dt \qquad (5\text{-}19)$$

which roughly approximates the weighting scheme used for the controller derived from the digital model. Two strategies yield similar results.

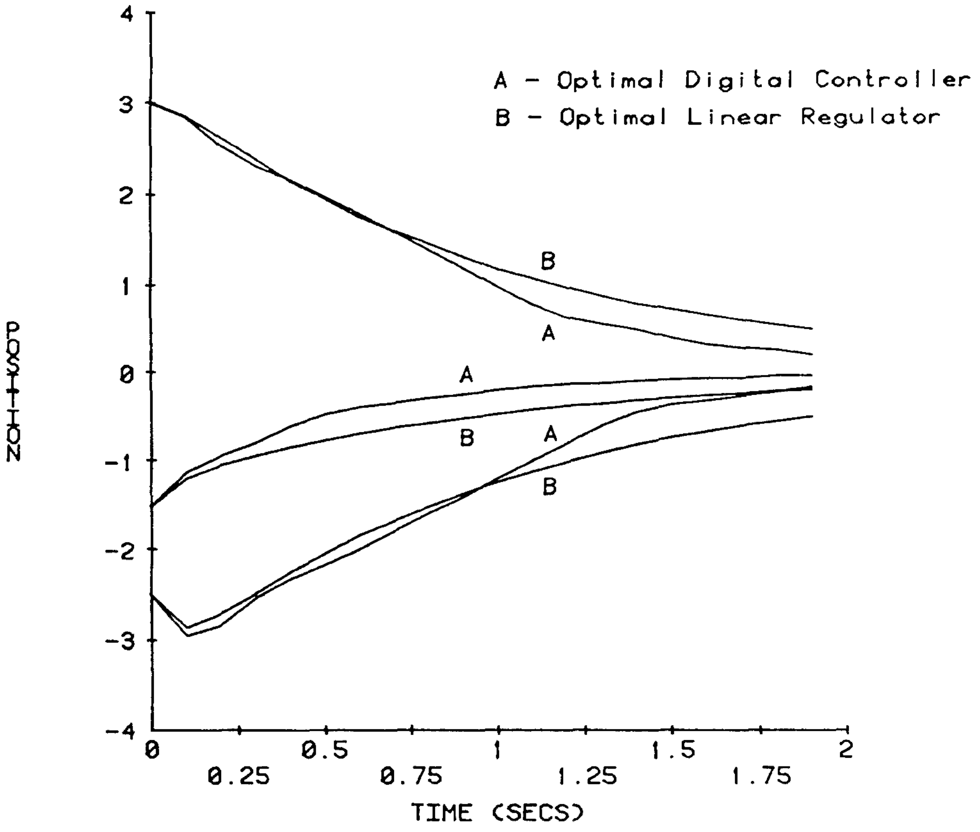


Figure 5.6. Simulated System

5.2. User Manual for the Program "DTQD"

5.2.1. Introduction.

The program "DTQD" is an aid in developing controllers for discrete-time quantized data (DTQD) systems.

"DTQD" is a menu-driven program with a hierarchial structure. It is divided into six basic parts, each being described in the following sections. A command level, the main menu, is used to access each of the five other levels: initialization, pararmeter modification, display, control law development, and simulation.

The program is coded in PL1. It was designed to be run on Honeywell 68-DPS-2 MULTICS system computers, but without serious modifications could be implemented on any system. The user need only type "DTQD" to execute the program. The program starts out at the initialization sub-level and proceeds to the command level after the job has been designated. From that point on, the user has control and may go to any of the five sub-levels.

The status of the current "job" is monitored by five flags. When a flag is set it indicates that that part of the job has been developed. The five flags designate whether or not a continuous-time and/or a discrete-time model of the system exists, whether a quantized model exists, if a control law has been developed, and whether or not a file containing a simulation of the controlled system has been made.

During the run, the user may be asked to input three different types of responses: a yes/no answer, a number from a multiple choice menu, and numerical data. If a yes/no answer is required, the following are acceptible answers: "y", "yes", or "n", "no". If a choice from a menu is requested, only an integer is considered a proper response. Finally, when inputting a numerical piece of data, only numbers, decimal points, and minus signs are acceptible. In case of a mistake, MULTICS allows a "#" sign to "erase" the previous character inputted, and a "@" sign to "erase" the entire line.

5.2.2. Initialization.

The first menu displayed is the initialization menu. It gets the user to open a data file. This may be a new file, an old file, or the user may wish to take the data from an old file and copy it into a new file and work with the new file. The data file is referenced by a "job" name. This name may be any one word with a maximum length of 50 characters, and is inputted by the user. It may be any combination of numbers, letters, and underscores; however, the first character must be a letter ("$" is considered to be a letter) and the name may not contain blanks or periods (.). The job name will also reference all other files made concerning the job: the quantized data file, also called the next-state file, which contains the coded version of how each state is affected by each input (See section 5.2.4.13. for coding procedure), the control law, and the simulation file. The initalization menu appears as follows:

```
1. Access an old job file
2. Create a new job file                    (init. menu)
3. Modify an old job file
4. Quit
```

5.2.2.1. Open an old file. If a "1" is entered, the user is prompted to enter a job name. The data file job_name.DATA is accessed. If the flag which monitors the existence of a quantized model is set, but a file containing the model does not exist, the program proceeds to automatically build the next state array. If, however, the quantized model does exist in a file, that file is accessed in addition to the data file. Similarly, if the control flag is set, the program accesses the file containing the control law. After completing this process, the Main Menu appears and the user is at the Command level.

5.2.2.2. Open a new file. The program prompts the user for a job name and then a title for the data file. The title may consist of up to 70 characters. However, if it is made up of more than one word it needs to be entered within quotation marks ("). The user is then sent to the parameter modification level. (See section 5.2.4.) At that level the user is prompted to enter any/all of the parameters concerning the model of the system and the A/D converter. After the models have been built the user is sent to the Command level.

5.2.2.3. Copying an old file into a new file. By entering a "3" the user is able to access an old file, copy the data file from it into a new file, and work with the new file. In this way the user may modify existing data and yet not destroy the original data. The user must enter the job name of the old file and then a new name for the new file. The program then proceeds as in case (1) above (accessing an old file) by building or opening the files containing the quantized model and the control law if the status flags are set.

5.2.2.4. Quit. If a "4" is entered it is assumed that the user does not want to initialize a new job, and the user is sent to the command level.

5.2.3. Command Level.

The Command Level is primarily the "main menu" which consists of the following options:

```
1. Initialize
2. Modify Data File
3. Print files
4. Develop Control Law
5. Simulate                                 (main menu)
6. Quit
```

5.2.3.1. Initialize. This level allows the user to choose a different job file to work with. (See section 5.2.2.) Thus, the user is essentially re-executing the program. Before re-starting the initialization process, all modifications to the current job are saved and the data file is closed.

5.2.3.2. Parameter modification. At this level the user is able to modify any of the parameters in the data file: the continuous-time, discrete-time, or quantized system parameters. (See section 5.2.4.)

5.2.3.3. Display. This response allows the user to examine other files (See section 5.2.5.) The display level is entered, and the user can look at the data in the job file as well as the next-state array, and/or the control law. The status of the job, and a summary of the quantization levels can also be examined.

5.2.3.4. Control law. This choice executes the control law development level (See section 5.2.6.)

5.2.3.5. Simulate. This selection simulates the controlled system (See section 5.2.7.)

5.2.3.6. Quit. This choice ends the program. If a quantized model of the system exists for the job, the user can save this model in a file. The status flag for the quantized model is not affected by this decision. If the data is not saved, then the next time the job is accessed, the quantized model will be automatically rebuilt instead of read in from the file. Finally, all files are closed and the program is exited.

5.2.4. Parameter Modification.

This level may be accessed via the command level or by the initialization level if a new job is created. The parameter modification level is made of three sub-levels, each accessing even further sub-levels. The user may enter or modify the continuous-time model for the system. The program can then generate a discrete-time model or allow the model to be entered by the user. Similarly, the quantized model may be generated or a file containing the quantized model may be accessed.

When the continuous system is modified, the discrete and quantized models are no longer valid and so their status flags are cleared. Similarly, the control law and simulations can no longer be associated with the model and their status flags are also cleared. This process is continued throughout the program: when a model or file is modified, all models and files generated from it are invalid and hence their status flags are cleared.

Upon entering this level, the following may be modified or created.

```
1. Title of the job file
2. Continuous system parameters
3. Discrete system parameters              (param menu)
4. Quantized system parameters
5. None of the above
```

5.2.4.1. Title. The user is prompted to enter a title. It can have a length of up to 70 characters; however, if it is more than one word it must be entered inside quotation marks (").

5.2.4.2. Continuous parameters. As soon as a continuous model of the system is created or modified, the status flags for the discrete-time and quantized models, the control law, and any simulations are cleared. If a continuous system already exists, the user may choose the parameters which need to be modified.

```
1. Number of states
2. Number of inputs
3. System matrix, A                     (param.2 menu)
4. Input matrix, B
5. All of the above
6. None of the above
```

This menu will continue to re-appear until a "6" is entered. If a continuous system does not currently exist, this menu does not appear; it is assumed that the user wishes to enter all of the parameters (i.e, that a "5" was entered).

5.2.4.3. Number of states. The user is asked to enter the number. It must be an integer and have a value no larger than ten. Since the number of states affects the dimensions of the system and input matricies, the user is also prompted to enter all of the components of each of these matricies. The above menu (param.2) is then re-displayed so that other changes may be made if desired.

5.2.4.4. Number of inputs. The number of inputs must be an integer. As in the above case, a change in the number of inputs will cause a change in the dimensions of the input matrix, B. For this reason, the user is then automatically asked to enter the entire B matrix.

5.2.4.5. System matrix. The program prompts the user to enter the A matrix. After entering all the components, the above menu (param.2) is again displayed.

5.2.4.6. Input matrix. As in the above case, the program prompts the user to enter each element of the input matrix, B.

5.2.4.7. Modify all. The user is prompted to enter all of the above parameters in the order in which they appear in the menu param.2. After entering the data, the menu is displayed, giving the user an opportunity to re-modify any of the new data in case a mistake was made.

5.2.4.8. Modify none. If a "6" is entered it is assumed that the user has completed all the desired modifications of the continuous-time system. The user is returned to the (param) menu and can modify or create another model of the system.

5.2.4.9. Discrete parameters. If the user wishes to modify or create the discrete-time parameters, a "3" should be entered when the menu (param) is displayed. Upon generating, creating, or modifying the discrete model, the quantized model, control law, and simulation status flags are deleted for the current job. If a continuous model of the system exists the user can:

```
1. Generate a discrete model from the continuous model
2. Modify the discrete model
3. Quit                                        (param.3a menu)
```

If, however, a continuous model does not currently exist, the following menu is displayed:

```
1. Create a continuous model first
2. Modify/Create a discrete model
3. Quit                                        (param.3b menu)
```

5.2.4.10. Discrete model generation. If a continuous - time model exists, the user is asked to enter the time constant tau, and then the program will automatically discretize the continuous model and display the new discrete-time system and input matricies. If, on the other hand, a "1" is entered when menu (param.3b) is displayed, the program will prompt the user for the continuous - time parameters. Thus, the user is sent to another level, and is then able to enter the continuous system.

5.2.4.11. Discrete model modification. A discrete-time model of the system may be entered independently from the continuous model. Caution: If this is done when the menu (param.3a) had been displayed, (i.e, when a continuous system exists) the user will be making the continuous model invalid since it will no longer represent the same system.

The user is asked to select the parameters to be modified.

```
1. Number of states
2. Number of inputs
3. Discrete system matrix                      (param.3.1 menu)
4. Discrete input matrix
5. All of the above
6. None of the above
```

This menu is very similar to menu (param.2) for the continuous-time case; thus, a description is omitted here.

5.2.4.12. Quit. This entry will cause the menu (param) to be displayed.

5.2.4.13. Quantized parameters. There are two methods of obtaining a quantized model of the system. When a new model is generated or accessed, the control law and simulation status flags are automatically cleared.

If a discrete model of the system exists, the user may generate a quantized system from the discrete model. The following parameters of the A/D converter must be entered: the number of quantization steps for each state, the upper and lower voltage bounds for each state, the number of quantization steps for each input, and the upper and lower voltage bounds for each input. If, on the other hand, the user does not want the quantized system generated, a separate file which already contains a quantized model of the system can be accessed. If a discrete model of the system does exist and the user accesses this file, the discrete model may no longer be valid.

If a discrete model does not exist, the user can create one or access a separate file containing a quantized model of the system. If it is desired to create a discrete model, the menu (param) appears. The user may then input a "3" and begin to generate or create a discrete-time model. If the user wishes to access a file containing the quantized system (i.e, a next-state file containing the affects of each input on each state), the program asks for the name of the data file.

After modifying, creating, or generating the quantized model of the system, the user can have the next-state array displayed. (See section 5.2.5.3.) The states and inputs are each coded. The codes are used throughout the program and, more importantly, are used to represent the states and inputs when printing out the next - state array, the control law, and cell status array. The states and inputs are coded in the following manner: the smallest possible state has a code of 1; the first state is increased to its next possible value and then coded with a 2; the first state continues to be incremented until it reaches its largest possible value minus one step. Next, the second state is incremented by one step and the process is repeated. The coding continues until all possible state combinations have been coded. The procedure for coding the inputs is similar.

5.4.4.14. Example 1: Coding the states and inputs. Assume that the user inputs the following A/D parameters:

number of states = 2;
number of inputs = 1
number of quantization steps for state 1 = 4
number of quantization steps for state 2 = 8
upper and lower voltage bounds for state 1 = 4, -4
upper and lower voltage bounds for state 2 = 2, -2
number of quantization steps for the input = 4
upper and lower voltage bounds for the input = 1, -1

Now, the program can code the states and inputs in the following manner:

number of state combinations = 4 x 8 = 32
  the step size for state 1 = (4 - (-4)) / 4 = 2
  the step size for state 2 = (2 - (-2)) / 8 = 0.5
number of input combinations = 4
  the step size for the input = (1 - (-1)) / 4 = 0.5

Thus, there are 32 state codes and 4 input codes.

The smallest possible state = $\begin{bmatrix} -4 \\ -2 \end{bmatrix}$; it has a state code of 1

Increasing state 1 by 1 step size = $\begin{bmatrix} -2 \\ -2 \end{bmatrix}$; it has a state code of 2

Similarly, the code for $\begin{bmatrix} 0 \\ -2 \end{bmatrix} = 3$, $\begin{bmatrix} 2 \\ -2 \end{bmatrix} = 4$.

Note the case of $\begin{bmatrix} 4 \\ -2 \end{bmatrix}$ is not included; the process codes the states from lower voltage level to the (upper voltage level - 1 step).

Next, the process is repeated after first incrementing the second state by one step. Thus $\begin{bmatrix} -4 \\ -1.5 \end{bmatrix}$ has a code of 5, $\begin{bmatrix} -2 \\ -1.5 \end{bmatrix}$ has a code of 6 ...
The process continues until finally, $\begin{bmatrix} 2 \\ 1.5 \end{bmatrix}$ has a code of 32.

The quantized model of the second order system may be thought of as a cell plane, with first state along the horizontal axis and the second state along the vertical axis. The two cell planes for this example (See Figure 2-1) graphically illustrate the discrete states and their codes. A similar process is used to code the inputs.

5.2.4.15. Quit. This selection returns the user to the command level.

5.2.5. Display Level.

At this level the user may choose to have any of the following displayed:

1. Status of the job
2. Data file
3. Quantized data (next-state) file
4. Summary of quantization levels
5. Control Law (display menu)
6. None of the above

The above menu may vary depending on the validity of the files. For example, if a control law does not exist yet for the job, choices 4 and 5 are omitted.

5.2.5.1. Check status. This option allows the user to see which representations of the system are valid: the continuous-time, discrete-time, and/or the quantized model. Also, two checks are made to see whether or not a control law exists for the job and if a file containing simulation data exists.

5.2.5.2. Data file. This choice tells the program to display the continuous-time, discrete-time, and the A/D converter parameters. (Note: at the present time this option does not work.)

5.2.5.3 Next-state array. The next-state array is two dimensional, and displayed such that the code for each state is on the vertical "axis" and

-4 1.5	-2 1.5	0 1.5	2 1.5
-4 1.0	-2 1.0	0 1.0	2 1.0
-4 0.5	-2 0.5	0 0.5	2 0.5
-4 0	-2 0	0 0	2 0
-4 -0.5	-2 -0.5	0 -0.5	2 -0.5
-4 -1	-2 -1	0 -1	2 -1
-4 -1.5	-2 -1.5	0 -1.5	2 -1.5
-4 -2	-2 -2	0 -2	2 -2

Grid 1

29	30	31	32
22	23	24	25
18	19	20	21
17	18	19	20
13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Grid 2

Figure 5-7. Quantization Grids

the code for each input is printed along the horizontal. Lying within the matrix are the codes representing the states to which the corresponding state would move, given the corresponding input. The coding procedure is discussed is section 5.2.4.13. If the code is a zero (0), it implies that the given state is saturated or leads to a uncontrollable cell. An uncontrollable cell is one which leads to a saturated state for all possible inputs. After printing the next state array for ten states the user is given the option to continue displaying the array. This question is asked after every ten states.

5.2.5.4. Example 2: Format of next-state arrays. The next state array is a two-dimensional array of dimension number of state combinations by number of input combinations. If the user enters the A/D parameters as described in Example 1, the first part of the quantized data array could appear as follows:

```
 1    9    9   17   17
 2   10   10   18   18
 3    0    0    0    0
 4   12   12   20   20
 .              .
 .              .
32   17   17    9    9
```

In this example, as in example 1 of section 5.2.4.14., there are 32 state combinations and 4 input combinations. The first row of the array tells the user that if the current state, x(k), has a code of 1 and an input is applied which has a code of 1 or 2, then the next state, x(k+1), will have a state code of 9. Similarly, if an input is applied whose code is 3 or 4, the next state's code will be 17. Any input will cause the third cell to saturate or become uncontrollable.

5.2.5.5. Check quantization level. This option allows the user to make some crude checks regarding the quantization. Two basic checks are done. The first is a summary of the cells moved from each state with a zero input. The number of cells moved in each direction and the total number of cells moved are computed and displayed.

The second part of the report checks the number of cells moved from the zero state for each input at its smallest value. If the smallest value results in saturation, the smallest value which results in a non-saturated next state is used. The results are reported for each input. In this part, unlike the first, the cell movement is described by an absolute and average value. The absolute value represents the number of cells moved in the given direction, while the average value is the ratio of the number of cells moved to the number of steps between the smallest non_saturating input and the zero input. These absolute and average values are recorded, as in the first part, for cell movements in each direction as well as the total number of cells moved. (See Example in section 5.2.5.6.)

After displaying the summary, the user has the option to have the saturation edge array printed. This array has the same matrix format as the next state array, but elements are displayed as either an "F" or a "T." A "T" is displayed if the cell leads to saturation or to an uncontrollable cell when the corresponding input is applied. For instance, if the beginning of the array appears as:

```
1    F   F   F   F
2    T   F   F   F
3    T   T   T   T
4    F   F   F   F
```

it implies that any input will cause the state whose code is 3 to lead to saturation or to an uncontrollable cell. This result also occurs if an input which has a code of 1 is applied to state code 2.

If some of the cells are uncontrollable, the user has the opportunity to print out the uncontrollable cell array. This array contains the codes of the cells which are uncontrollable.

5.2.5.6. Example 3: Calculation of cell movement. Using the second cell plane, Figure 5-7, assume that the state with a code of 10 moves to the state coded by 19 when the zero input is applied. The movement in the direction of the first state (horizontal movement) is 1 cell and the movement in the second direction is 2 cells. Thus, the total number of cells moved is 1 + 2 = 3 cells.

From Example 1 of 5.2.4.14., the zero state has a code of 19, and the zero input has a code of 3. Assume that the smallest input (-1 volts) leads the zero state into saturation but the next smallest input (-0.5 volts) leads the zero state to the state whose code is 15. The number of steps between the zero input and the minimum non-saturating input is one, since there is only one step between 0 and -0.5. The absolute movement then, is 0 cells in the direction of state 1 and is 1 cell in the direction of state 2. The average movement is 0/1 = 0 cells in direction 1, and 1/1 = 1 in the second direction.

5.2.5.7 Control law file display. The control law file for the job is printed. The display is an array giving the appropriate input code for every possible state combination to obtain the desired controller. (See section 5.2.4.14. for an explaination and example of the coding process.)

5.2.5.8 Quit. This choice causes the program to exit the display level and return to the command level.

5.2.6. Control Law Development.

Upon entering this level, the user is asked to enter the type of cost function to be used in developing the control law.

1. Minimum Time
2. Quadratic

3. Minimum Control Effort
4. Custom Cost Function (control menu)
5. None - Access a control law file
6. None of the above

5.2.6.1. Minimum time. The program attempts to build a control law which satisfies the requirements of a minimum time cost function. Thus, the controller will be one such that the control input will take the current state to the origin in the least amount of time.

5.2.6.2. Quadratic. If this cost function is chosen the user is asked to enter the state and input cost matricies (the "Q" and "R" matricies). These weighting matricies are assumed to be diagonal, so only the diagonal elements are needed.

5.2.6.3. Minimum control effort. As in the case above, the user is asked to enter the input weighting matrix ("R"). Again, this is assumed to be a diagonal matrix.

5.2.6.4. Custom cost function. If none of the above choices are desirable, the user may write a custom cost function. To do this, a procedure should be written in PL1 and named custom_cost_function. The discrete state and input arrays are passed to custom_cost_function and the procedure should compute and return the cost. All three parameters need to be declared as floating arrays/numbers.

5.2.6.5. Access a file. The user may choose to implement an already-developed control law by entering the name of the file so that the program can access it.

5.2.6.6. Quit. This is the correct choice if the user does not wish to build a control law, but does want to return to the command level.

After choosing the cost function (if a "6" was not chosen), the user is prompted to enter the center and edge cell tolerances. The center cell tolerance is used by the program to determine the tolerant region which surrounds the origin. Within this tolerant region, the program checks to see if any cells exist which can not reach the origin with any of the possible inputs, yet other cells which are also unable to get to the origin are able to reach them. These cells are called root cells. So, if a center cell tolerance of 1 is entered for the system discussed in Example 1, the program would check to see if any of the following cells were root cells: 14, 15, 16, 18, 20, 22, 23, and 24.

The edge cell tolerance is used to compensate for edge irregularities. If this tolerance is input to be 1, for the system described in 5.2.4.14., the edge cells would be: 1 - 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24, 25, 28, and 29 - 32. Both the center and edge cell tolerances must be entered as integers.

The program continues by attemting to build the cell status array; it finds all root cells and the cells which are reachable to them. If successful, the tolerant region is built. If the tolerant region control law can be constructed, the program then builds the control law and sets the control law status flag.

If the status flag for the control law is set, the user can have the cell status array and control law printed. The cell status array is an array which codes each state in the following manner:

```
  0:  Unmarked cell
  1:  Cell is uncontrollable
  2:  Cell is in the edge tolerant region
  3:  Root Cell - the zero state cell
  4:  Cells which can reach the Root cell coded with a 3
  5:  Root Cell
  6:  Cells which can reach the root cell coded with a 5
  7:  Root cell
  8:  Cells which can reach the root cell coded with a 7
  .
  .
  i:  Root Cell
i+1:  Cells which can the root cell coded with an "i"
```

The control law is printed out just as in the Display level (See section 5.2.5.7.) The appropriate input code which has been found to satisfy the chosen cost function is printed for each state code.

5.2.7. Simulation Level.

After the control law has been developed for the job, the user may wish to simulate the controlled system. To simulate the system, the program calls an IMSL routine, DVERK, which solves the system of differntial equations or OWN_SYS_TO_SIM if a system other than the one in the job file is to be simulated. A simulation of the system may only be obtained after the parameters for the quantized system have been entered and a control law has been developed.

Upon entering this level, the following menu or question is displayed, depending whether or not a simulation file exists for the current job:

```
If a simulation does not exist:
     Would you like to simulate the system?

If a simulation file does exist:
     Would you like to:

     1.  Modify the simulated data file
     2.  Plot the existing simulated data        (Sim. menu)
     3.  Quit
```

5.2.7.1. Simulating. This response lets the user start the simulation process. The program then gives the user various parameters needed for the simulation. First, the user can have any continuous model of the system, not necessarily the one in the data file, be simulated. This is desirable if the user wants to see how the control law works on slightly permutated systems. With this option, the user can take a nonlinear system, find a linear representation of it and use DTQD to develop the control law, and then simulate the nonlinear model using this control law. If this is desired, the user must write a PL1 routine, and name it own_sys_to_sim.pl1. Note: the states to be accessed by the control law must be the first states in the system of equations. This limitation implies that the number of equations in own_sys_to_sim be equal or greater than the number of states used in the development of the job file. The procedure own_sys_to_sim should have the following parameters:

num_of_equations - fixed binary (35) - the number of simultaneous differential equations to be solved (i.e. the number of states);
time - float binary - the current time;
time_end - float binary - the time after doing subroutine;
state - (10) float binary - the state array upon entering routine;
state_after - (10) float binary - the state array after subroutine;
time_init - float binary - the initial time for the entire simulation;
time_end - float binary - the final time of the simulation;

It should also call a subroutine which will determine the next state. (e.g. DVERK) Whether or not the user accesses a separate file, the user is asked to enter the number of steps per time constant. This number should be an integer and not zero. At each step the program will call IMSL_DVERK or OWN_SYS_TO_SIM and have the next state determined. In this way the continuous-time model is simulated and the user can observe what is happening between sampling intervals. The number of embedding levels must be entered next. This value should also be a integer. The number of embedding levels is the number of times the controller is allowed to "zoom in." A zero (0) should be entered if the user does not want to access any other levels. If an integer other than zero is entered, the user is asked to enter the scaling factor. This value should be greater than zero and less than or equal to one. The program progresses to a different region, j, whenever the state is less than (the upper bound for the state) x (scale factor)^j, or greater than (the lower bound for the state) x (scale factor)^j. After the region is determined, the control law is accessed such that each of the control law inputs are also "scaled down" into the appropriate region. (See section 5.2.7.2. for an example)

Next, the user can have the simulated data displayed while running. A response of "yes" causes all the simulation data, time, states, and control inputs, to be printed on the screen. If at any time during the simulation one or more of the states becomes greater than its upper bound or less than its lower bound, the simulation is ended and a warning appears to let the user know that system has gone unstable. Whether the simulation is successful or not, the user can save the simulated data in a file and plot the data. If the data is saved in a file, it may opened later to study the data. If a file is made, the simulation status flag is set.

5.2.7.2. Example 4: Recursion levels and scaling factor. Using the A/D parameters of Examples 1 and 2, recall that in the previous examples, the voltage bounds for each state were as follows:

upper and lower voltage bounds for state 1 = 4,-4
upper and lower voltage bounds for state 2 = 2,-2

If the user enters "3" for the number of recursion levels, and 0.1 for the step size, the program will "zoom in" whenever the first state becomes smaller in magnitude than (0.1 x 4) = 0.4, or when the second state becomes smaller in magnitude than (0.1 x 2) = 0.2. If the states become smaller in magnitude than 0.04 or 0.02, respectively, the controller will zoom in a second time. If state 1 had a value of 0.3, smaller than 0.4 but greater than 0.04, the state would be at the first level. The control law would be accesssed as if the state had a value or 3 instead of 0.3, the control input would be found, and then scaled down to size. Thus, if the control law listed 5 volts as the proper input for a state of 3, the input that would be used would be 0.5 volts.

5.2.7.3 Plotting. If a simulation file exists for the job, a plot can be made immediately after entering the simulation level. Otherwise, the plot can be made following a simulation. Several parameters must be entered if the user wishes to make a plot. The user can make several plots on top of one another. Also, any state, any input, or the time can be plotted on either axis. The user may have any ascii keyboard character symbolize each data point or may opt to have no symbols at all. If symbols are used, the user may or may not choose to have them connected by vectors. In addition, the user may have the graph made with tick marks, a dotted grid, or a solid grid. Also, a title and axis labels may be entered. These labels have a maximum length of 25 characters. Finally, the user may have the program automatically scale the plot or opt to choose and enter the upper and lower bounds for each axis.

5.2.7.4. Quit. The user returns to the command level if this choice is selected.

5.2.8. An Overall Example.

As an example, consider a d.c. servomotor. To find a control for the motor, the program DTQD could be implemented as follows. First, a linear model of the system must be developed to represent the motor. For this example, we will use the following second order system as the model:

$$\dot{\underset{\sim}{x}}(t) = \begin{bmatrix} \dot{\Theta}(t) \\ \ddot{\Theta}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} \Theta(t) \\ \dot{\Theta}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \qquad (5\text{-}20)$$

DTQD can now be used to determine the discrete-time model using the above model and a chosen time constant τ. (The following pages contain the actual program run.) With $\tau = 0.25$ sec^{-1}, the discrete-time model was found to be:

$$x(k+1) = \begin{bmatrix} 1.00 & 0.20 \\ 0.00 & 0.61 \end{bmatrix} \begin{bmatrix} \Theta(k) \\ \Theta(k+1) \end{bmatrix} + \begin{bmatrix} 0.05 \\ 0.39 \end{bmatrix} u(k) \qquad (5\text{-}21)$$

From this model, the user can make a quick estimate of how state 1 and 2 are related by looking at the state trajectories. Assume that the voltage bounds are ±4 volts for each state and ±10 volts for the input. If $x(0) = \begin{bmatrix} \Theta(0) \\ \Theta(1) \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$, then from (5-21) $x(1) = \begin{bmatrix} 0.80 \\ 2.44 \end{bmatrix}$. The initial cell moved 0.8 cells in the direction of the first state and 1.6 cells in the direction of the second state (a 1:2 ratio). Therefore, an estimate for an average cell movement can be made. Using the 1:2 ratio as a guide and the voltage limits, an estimate can be made regarding the number of steps needed for quantizing the state and input.

Using 16 steps for the first state, 8 steps for the second and 8 for the input, DTQD can be implemented to determine the quantized model. Next, the user may opt to have DTQD breifly summarize the quantization and cell movement. From this summary, the user can determine whether of not the initial estimate for the number of steps was satisfactory.

After the user is satisfied with the quantized model, a control law for the system can be developed. In this example a minimum time cost function was chosen. Finally, the system may be simulated.

The following pages contain the program run for this example. An exclaimation point (!) before a word or number implies that the entry was input by the user.

```
! DTQD

  Would you like to :

  1.   Access an old job file
  2.   Create a new job file
  3.   Modify an old job file
  4.   Return to Main Menu

  Please choose one of the above => ! 2

  Enter name of the new job file => ! servo_motor


  Enter a title for the data file
  Note:  Quotes are required if more than one word is used
! "This is an EXAMPLE"


 Which of the following would you like to modify/create?

 1.   Title of the job file
 2.   Continuous system parameters
```

```
3.   Discrete system parameters
4.   Quantized system parameters
5.   None of the above

 Please choose one => ! 2


Enter number of states => ! 2


Enter number of inputs => ! 1


Enter values for the A matrix
    A (  1,   1) => ! 0
    A (  1,   2) => ! 1
    A (  2,   1) => ! 0
    A (  2,   2) => ! -2


Enter values for the B matrix
     B (  1,   1) => ! 0
     B (  2,   1) => ! 1


Which parameter(s) would you like to change?
1.   Number of states
2.   Number of inputs
3.   System matrix, A
4.   Input matrix, B
5.   All of the above
6.   None of the above

Please choose one => ! 6


Which of the following would you like to modify/create?

1.   Title of the job file
2.   Continuous system parameters
3.   Discrete system parameters
4.   Quantized system parameters
5.   None of the above

 Please choose one => ! 3


A continuous model exists, would you like to:

1.   Generate a discrete model from the continuous system
2.   Enter a new discrete system
3.   Quit

Please choose one => ! 1

     Enter tau =>! 0.25


PHI MATRIX =
 1.00000000e+000     1.96734663e-001
 0.00000000e+000     6.06530674e-001

LAMBDA MATRIX =
 5.32653360e-002
 3.93469326e-001
```

```
  Which of the following would you like to modify/create?

  1.   Title of the job file
  2.   Continuous system parameters
  3.   Discrete system parameters
  4.   Quantized system parameters
  5.   None of the above

   Please choose one => ! 5


  1) Initialize
  2) Modify Data File
  3) Print Files
  4) Build Control Law
  5) Simulate
  6) Quit

  Enter choice  ==>  ! 6
  r 15:02 2.693 132

! DTQD

  Would you like to :

  1.   Access an old job file
  2.   Create a new job file
  3.   Modify an old job file
  4.   Return to Main Menu

 Please choose one of the above => ! 1

 Enter the job name => ! servo_mctor

 The title of this data file is:
    This is an EXAMPLE
 Is this the correct file?  ! y

 The current status of this job is:

      A continuous system exists
      A discrete system exists

 1) Initialize
 2) Modify Data File
 3) Print Files
 4) Build Control Law
 5) Simulate
 6) Quit

 Enter choice  ==>  ! 2


 Which of the following would you like to modify/create?

 1.   Title of the job file
 2.   Continuous system parameters
 3.   Discrete system parameters
 4.   Quantized system parameters
 5.   None of the above

  Please choose one => ! 4


 Would you like to generate a quantized system
            from the discrete system? => ! y
```

```
Enter the number of quantization steps
   for state number     1 => ! 8
   for state number     2 => ! 4

Enter the upper and lower voltage bounds (u, l)
   for state number     1 => ! 4,-4
   for state number     2 => ! 4,-4

Enter the number of quantization steps
   for input number     1 => ! 4

Enter the upper and lower voltage bounds (u, l)
   for input number     1 => ! 10,-10


Would you like the next state file built! y


     Building next state file
Would you like the next state file printed? => ! n


Which of the following would you like to modify/create?

1.   Title of the job file
2.   Continuous system parameters
3.   Discrete system parameters
4.   Quantized system parameters
5.   None of the above

 Please choose one => ! 5



1) Initialize
2) Modify Data File
3) Print Files
4) Build Control Law
5) Simulate
6) Quit

Enter choice   ==>   ! 3



Which of the following would you like printed?

1.   Status of the job
2.   Data file for the job
3.   Quantized data file - next state file
4.   Quantization level check
5.   None of the above

Enter choice => ! 4

Would you like to check the quantization level ? ! y

Number of controllable cells =                        32
    Total number of cells   =                         32

         Number of cells moved in each direction
Dir      0    1
    1    24    7
    2    15   16

         Number of cells moved total
num      0    1
         8   23
```

```
Input     Input       Num    Total Cells          Cells Moved
          Status      Input    Moved                  in
                      Steps  Abs    Avg         Dir    Abs     Avg
  1     max unsat       2      2     1.00
                                                 1      0       0.00
                                                 2      2       1.00

Would you like the saturated edge array printed? ! n


Which of the following would you like printed?

1.  Status of the job
2.  Data file for the job
3.  Quantized data file - next state file
4.  Quantization level check
5.  None of the above

Enter choice => ! 5

1) Initialize
2) Modify Data File
3) Print Files
4) Build Control Law
5) Simulate
6) Quit

Enter choice  ==>  ! 2


Which of the following would you like to modify/create?

1.  Title of the job file
2.  Continuous system parameters
3.  Discrete system parameters
4.  Quantized system parameters
5.  None of the above

 Please choose one => ! 4


A quantized system currently exists
Do you still wish to modify the quantized system? => ! y

Would you like to generate a quantized system
           from the discrete system? => ! y

Enter the number of quantization steps
   for state number     1 => ! 16
   for state number     2 => ! 8

Enter the upper and lower voltage bounds (u, l)
   for state number     1 => ! 4,-4
   for state number     2 => ! 4,-4

Enter the number of quantization steps
   for input number     1 => ! 8

Enter the upper and lower voltage bounds (u, l)
   for input number     1 => ! 10,-10

Would you like the next state file built! y


    Building next state file
Would you like the next state file printed? => ! y
```

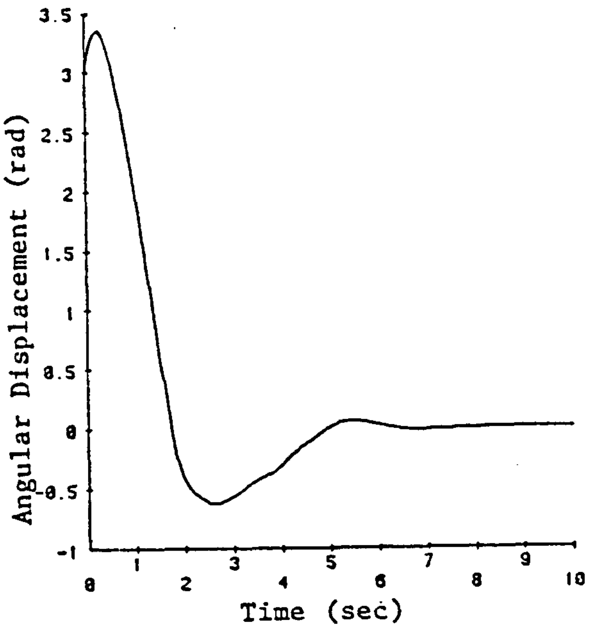
```
    1        0        0        0        0        0        0        0       81
    2        0        0        0        0       33       49       65       82
    3        0        0        1       17       34       50       66       83
    4        0        0        2       18       35       51       67       84
    5        0        0        3       19       36       52       68       85
    6        0        0        4       20       37       53       69       86
    7        0        0        5       21       38       54       70       87
    8        0        0        6       22       39       55       71       88
    9        0        0        7       23       40       56       72       89
   10        0        0        8       24       41       57       73       90
More? =>     ! y

   11        0        0        9       25       42       58       74       91
   12        0        0       10       26       43       59       75       92
   13        0        0       11       27       44       60       76       93
   14        0        0       12       28       45       61       77       94
   15        0        0       13       29       46       62       78       95
   16        0        0       14       30       47       63       79       96
   17        0        0        0        0        0        0       65       81
   18        0        0       17       17       33       49       66       82
   19        0        1       18       18       34       50       67       83
   20        0        2       19       19       35       51       68       84
More? =>     ! y

   21        0        3       20       20       36       52       69       85
   22        0        4       21       21       37       53       70       86
   23        0        5       22       22       38       54       71       87
   24        0        6       23       23       39       55       72       88
   25        0        7       24       24       40       56       73       89
   26        0        8       25       25       41       57       74       90
   27        0        9       26       26       42       58       75       91
   28        0       10       27       27       43       59       76       92
   29        0       11       28       28       44       60       77       93
   30        0       12       29       29       45       61       78       94
More? =>     ! y

   31        0       13       30       30       46       62       79       95
   32        0       14       31       31       47       63       80       96
   33        0        0        0        0       49       65       81       97
   34        0        1       17       33       50       66       82       98
   35        0        2       18       34       51       67       83       99
   36        0        3       19       35       52       68       84      100
   37        0        4       20       36       53       69       85      101
   38        0        5       21       37       54       70       86      102
   39        0        6       22       38       55       71       87      103
   40        0        7       23       39       56       72       88      104
More? =>     ! y

   41        0        8       24       40       57       73       89      105
   42        0        9       25       41       58       74       90      106
   43        0       10       26       42       59       75       91      107
   44        0       11       27       43       60       76       92      108
   45        0       12       28       44       61       77       93      109
   46        0       13       29       45       62       78       94      110
   47        0       14       30       46       63       79       95      111
   48        0       15       31       47       64       80       96      112
   49        0        0        0       49       65       81       97      114
   50        1       17       33       50       66       82       98      115
More? =>     ! y

   51        2       18       34       51       67       83       99      116
   52        3       19       35       52       68       84      100      117
```

```
  53    4   20   36   53   69   85  101  118
  54    5   21   37   54   70   86  102  119
  55    6   22   38   55   71   87  103  120
  56    7   23   39   56   72   88  104  121
  57    8   24   40   57   73   89  105  122
  58    9   25   41   58   74   90  106  123
  59   10   26   42   59   75   91  107  124
  60   11   27   43   60   76   92  108  125
More? => ! y

  61   12   28   44   61   77   93  109  126
  62   13   29   45   62   78   94  110  127
  63   14   30   46   63   79   95  111  128
  64   15   31   47   64   80   96  112    0
  65    0   17   33   49   65   82   98  114
  66    1   18   34   50   66   83   99  115
  67    2   19   35   51   67   84  100  116
  68    3   20   36   52   68   85  101  117
  69    4   21   37   53   69   86  102  118
  70    5   22   38   54   70   87  103  119
More? => ! y

  71    6   23   39   55   71   88  104  120
  72    7   24   40   56   72   89  105  121
  73    8   25   41   57   73   90  106  122
  74    9   26   42   58   74   91  107  123
  75   10   27   43   59   75   92  108  124
  76   11   28   44   60   76   93  109  125
  77   12   29   45   61   77   94  110  126
  78   13   30   46   62   78   95  111  127
  79   14   31   47   63   79   96  112  128
  80   15   32   48   64   80    0    0    0
More? => ! y

  81   17   33   49   65   82   98  114    0
  82   18   34   50   66   83   99  115    0
  83   19   35   51   67   84  100  116    0
  84   20   36   52   68   85  101  117    0
  85   21   37   53   69   86  102  118    0
  86   22   38   54   70   87  103  119    0
  87   23   39   55   71   88  104  120    0
  88   24   40   56   72   89  105  121    0
  89   25   41   57   73   90  106  122    0
  90   26   42   58   74   91  107  123    0
More? => ! y

  91   27   43   59   75   92  108  124    0
  92   28   44   60   76   93  109  125    0
  93   29   45   61   77   94  110  126    0
  94   30   46   62   78   95  111  127    0
  95   31   47   63   79   96  112  128    0
  96   32   48   64   80    0    0    0    0
  97   33   49   66   82   98   98  115    0
  98   34   50   67   83   99   99  116    0
  99   35   51   68   84  100  100  117    0
 100   36   52   69   85  101  101  118    0
More? => ! y

 101   37   53   70   86  102  102  119    0
 102   38   54   71   87  103  103  120    0
 103   39   55   72   88  104  104  121    0
 104   40   56   73   89  105  105  122    0
 105   41   57   74   90  106  106  123    0
 106      42     58     75     91    107    107    124      0
 107      43     59     76     92    108    108    125      0
 108      44     60     77     93    109    109    126      0
 109      45     61     78     94    110    110    127      0
 110      46     62     79     95    111    111    128      0
More? => ! y

 111      47     63     80     96    112    112      0      0
 112      48     64      0      0      0      0      0      0
 113      33     50     66     82     99    115      0      0
 114      34     51     67     83    100    116      0      0
 115      35     52     68     84    101    117      0      0
 116      36     53     69     85    102    118      0      0
 117      37     54     70     86    103    119      0      0
 118      38     55     71     87    104    120      0      0
 119      39     56     72     88    105    121      0      0
 120      40     57     73     89    106    122      0      0
More? => ! y

 121      41     58     74     90    107    123      0      0
 122      42     59     75     91    108    124      0      0
 123      43     60     76     92    109    125      0      0
 124      44     61     77     93    110    126      0      0
 125      45     62     78     94    111    127      0      0
 126      46     63     79     95    112    128      0      0
 127      47     64     80     96      0      0      0      0
 128      48      0      0      0      0      0      0      0


Which of the following would you like to modify/create?

1.   Title of the job file
2.   Continuous system parameters
3.   Discrete system parameters
4.   Quantized system parameters
5.   None of the above

 Please choose one => ! 5


1) Initialize
2) Modify Data File
3) Print Files
4) Build Control Law
5) Simulate
6) Quit

Enter choice  ==>  ! 3


Which of the following would you like printed?

1.   Status of the job
2.   Data file for the job
```

```
3.  Quantized data file - next state file
4.  Quantization level check
5.  None of the above

Enter choice => ! 4

Would you like to check the quantization level ? ! y

Number of controllable cells =                    128
    Total number of cells  =                      128

       Number of cells moved in each direction
Dir    0    1    2
   1   48   60   14
   2   46   61   15

       Number of cells moved total
num    0    1    2    3
      16   62   15   29

Input       Input      Num    Total Cells        Cells Moved
            Status     Input     Moved               in
                       Steps   Abs    Avg        Dir    Abs     Avg
   1    max unsat        4       5     1.25
                                                  1      1        0.25
                                                  2      4        1.00

Would you like the saturated edge array printed? ! n



Which of the following would you like printed?

1.  Status of the job
2.  Data file for the job
3.  Quantized data file - next state file
4.  Quantization level check
5.  None of the above

Enter choice => ! 5

1) Initialize
2) Modify Data File
3) Print Files
4) Build Control Law
5) Simulate
6) Quit

Enter choice  ==>  ! 4


Would you like to build the control law file? ! y

Which type of cost function would you like to use ?

    1) Minimum Time
    2) Quadratic
    3) Minimum Control Effort
    4) Custom Cost Function (use procedure custom_cost_functio .pl1
    5) None - Would like to access a control law file
    6) None of the above

Please choose one =>  ! 1


Enter the center cell tolerance => ! 2
Enter the edge cell tolerance => ! 2

          Tree sucessfully completed
     Sucessfully built tolerant region control law
     Building control law
Would you like the cell status array printed ? ! n

Would you like the control law printed? => ! y

     Control Law:

          1       8
          2       6
          3       8
          4       6
          5       8
          6       6
          7       8
          8       6
          9       8
         10       7
         11       4
         12       3
         13       3
         14       3
         15       3
         16       3
         17       7
         18       6
         19       7
         20       6
         21       7
         22       6
         23       7
         24       6
         25       7
         26       3
         27       2
         28       2
         29       2
         30       2
         31       2
         32       2
         33       5
         34       8
         35       5
         36       8
         37       5
         38       8
         39       5
         40       8
         41       6
         42       3
         43       2
         44       2
         45       2
         46       2
         47       2
         48       2
         49       8
         50       7
         51       3
         52       7
         53       8
         54       7
         55       8
```

56	7
57	5
58	2
59	1
60	1
61	1
62	1
63	1
64	1
65	7
66	6
67	7
68	6
69	7
70	6
71	7
72	6
73	5
74	1
75	1
76	1
77	1
78	1
79	1
80	1
81	6
82	5
83	6
84	5
85	6
86	5
87	6
88	5
89	4
90	1
91	2
92	1
93	1
94	1
95	1
96	1
97	5
98	7
99	5
100	7
101	5
102	7
103	5
104	3
105	1
106	1
107	1
108	1
109	1
110	1
111	1
112	1
113	5
114	5
115	5
116	5
117	5
118	5
119	2
120	3
121	1
122	1

```
     123      1
     124      1
     125      1
     126      1
     127      1
     128      1
1) Initialize
2) Modify Data File
3) Print Files
4) Build Control Law
5) Simulate
6) Quit

Enter choice  ==>  ! 5

Would you like to simulate the system? => ! y

Would you like to simulate:
     1.  The continuous system in the job file
     2.  A continuous system in another file

Please choose one =>  ! 1

Enter number of steps per time constant  => ! 5

Enter the number of recursion levels => ! 3

Enter the scaling factor => ! 0.2

Enter initial state
     initial state   1 => ! 3
     initial state   2 => ! 3

Enter initial time => ! 0

Enter final time => ! 10

Would you like the simulation printed while running => ! n

     Simulating system

Would you like to save the simulated data in a file? => ! n

Would you like to plot the simulated data? => ! y

Would you like multiple plots on one graph? ! n

What would you like to plot on the y axis?
1.  A state
2.  An input
3.  Time

Please choose one => ! 1

Which state do you wish to plot on the y axis? ! 1

What would you like to plot on the x axis?
1.   A state
2.   An input
3.   Time

Please choose one => ! 3

Would you like a symbol to represent each data point? => ! n

The graph will have tick marks, be automatically scaled,
     and have no labels
Would you like to change any of these default options? => ! n
```



```
Would you like to plot the simulated data? => ! n


1) Initialize
2) Modify Data File
3) Print Files
4) Build Control Law
5) Simulate
6) Quit
Enter choice  ==>  ! 6

Would you like to save the quantized state file? => ! n

r 14:36 0.070 0
```

5.3. Modelling the Elevation Stabilization System

5.3.1. Introduction.

The following sections describe the process of modelling the elevation stabilization system of the M-60 tank and determining the parameters needed to implement the controller design program, "DTQD."

The first step in the process involved modelling the gun and hydraulic servo system to obtain the open loop transfer function. The models were simplified to develop three representations of the system: two first order approximations, one with the trunnion damping modelled as viscous friction and the other with it modelled as coulomb friction, and a third order approximation.

In modelling the system, several assumptions were made. First, the gun was considered to have only a single degree of freedom, and the gunner was not included as part of this model. It was later assumed that because the distance between the trunnion and the mass center of the gun was relatively small compared to the length of the gun, for small angular velocities the velocity and acceleration of the mass center and trunnion could be considered equal. In linearizing the gun model, a first order Taylor series approximation was done. The nominal values for the angular velocity of the gun and hull acceleration was considered to be zero. Finally, the model of the fluid flow relationship in the hydraulic system was taken directly from manufacturer specifications.

Using the program "DTQD," the control law was developed for the first order system in which the trunnion damping was modelled as viscous friction. The three models of the system were then simulated using this control law.

5.3.2. Modelling the gun.

5.3.2.1. Gun Kinematics. From the dimensions of the gun, the kinematics could be analyzed. (See Figure 5-8) The relationship between all the necessary "angles" and "sides" were determined using some simple trigonemetric identities.

The angle Θ can be expressed in terms of the length of the actuator ℓ using the law of cosines.

$$c = \text{sqrt}\,[\,(4.5)^2 + (14.1)^2\,] = 14.8 \tag{5-22}$$

$$\ell^2 = (38.28)^2 + (14.8)^2 - 2(38.28)(14.8)\cos\Theta \tag{5-23}$$

Thus,
$$\cos\Theta = \frac{\ell^2 - (38.28)^2 - (14.8)^2}{2(38.28)(14.8)} \tag{5-24}$$

The angle ψ can be found in terms of Θ and ℓ using the law of sines:

$$\sin\psi = (\sin\Theta)(14.8/\ell) \tag{5-25}$$

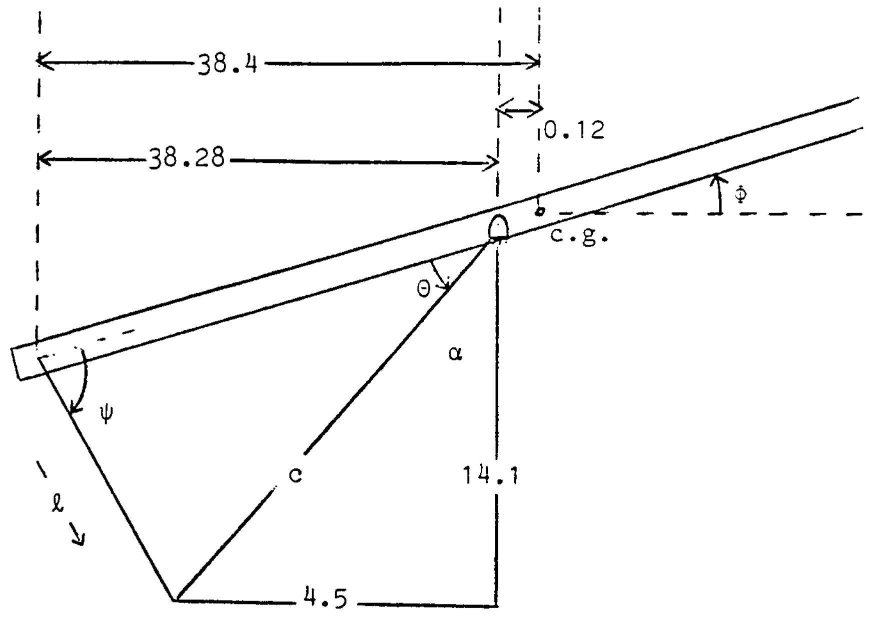


Figure 5-8. Model for deriving kinematics of gun

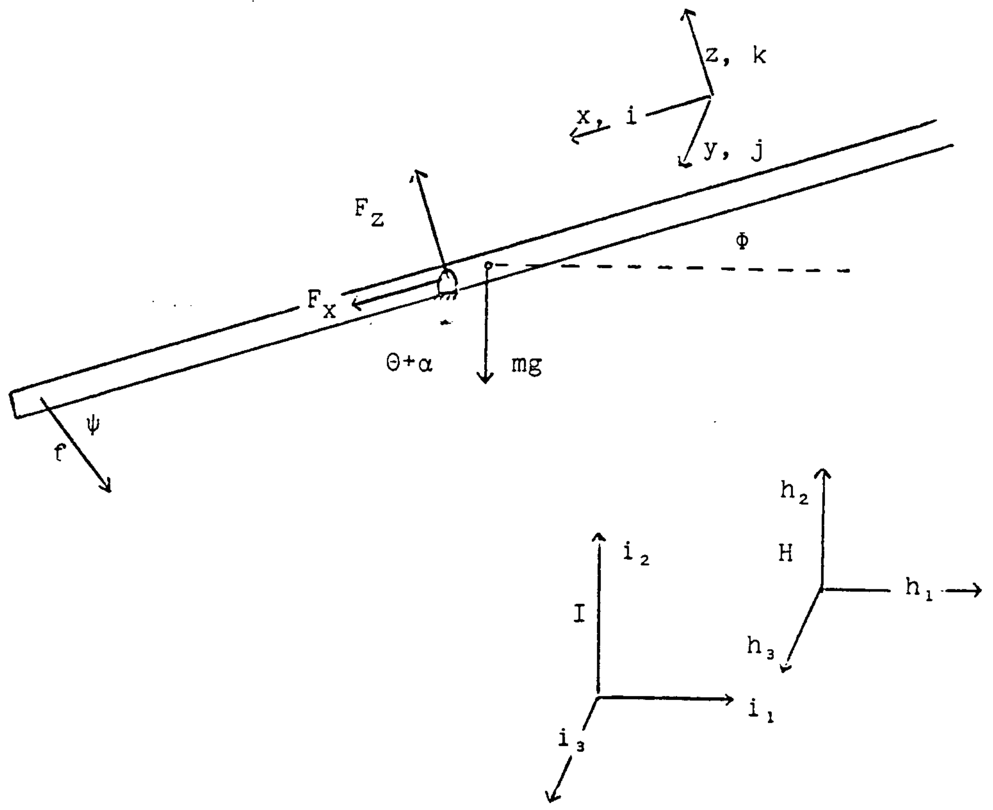


Figure 5-9. Model for deriving dynamics of gun

Finally, the angle Φ can be determined in terms of Θ as follows:

$$\alpha = \arctan [\ 4.5/14.1\ ] = 0.309 \text{ rad} = 17.7^\circ \tag{5-26}$$

$$\Phi = 180 - \Theta - \alpha - 90 = (1.26 - \Theta) \text{ rad} = (72.3^\circ - \Theta^\circ) \tag{5-27}$$

5.3.2.2. Gun Dynamics. The gun is treated as a rigid beam supported at the trunnion, and considered to have a single degree of freedom. To formulate the dynamic equations of the gun, define three coordinate systems: "H", the coordinate system fixed in the hull of the tank; "B", the body fixed system with the coordinates being the principal axis of the gun; and "I" the inertial reference frame. (See Figure 5-9) The angular coordinates are Φ for the gun, and β for the hull. Thus, the angular velocities may be expressed as:

$^{H}\vec{\omega}^{B}$ = angular velocity of the gun wrt. the hull = $\dot{\Phi}\ \vec{j}$

$^{I}\vec{\omega}^{H}$ = angular velocity of the hull wrt. the inertial ref. frame = $\dot{\beta}\ \vec{j}$

Assuming that the forces on the gun due to the hull are F_X and F_Z, and that the force due to the linear actuator is f, the equation for linear motion can be written as follows:

$$F_X\ \vec{i}\ +\ F_Z\ \vec{k}\ +\ f\ \vec{e}_1\ -\ mg\ \vec{i}_2\ =\ m\ ^{I}\vec{a}^{B} \tag{5-28}$$

where,

m = mass of the gun,

g = acceleration due to gravity,

$^{I}\vec{a}^{B}$ = acceleration of the gun with respect to the inertial reference frame, and

the coordinate $\vec{i}_2$ of the "I" reference frame and the coordinate $\vec{e}_1$ can be written in terms of the "B" reference frame by,

$$\vec{e}_1\ =\ -\cos\psi\ \vec{i}\ -\ \sin\psi\ \vec{k}, \tag{5-29}$$

$$\vec{i}_2\ =\ \cos(\Theta + \alpha)\ \vec{i}\ -\ \sin(\Theta + \alpha)\ \vec{k}. \tag{5-30}$$

Thus, equation (5-28) can be separated into two equations,

$$F_X\ -\ f\cos\psi\ +\ mg\cos(\Theta + \alpha)\ =\ m\ a_X \tag{5-31}$$

$$F_Z\ -\ f\sin\psi\ -\ mg\sin(\Theta + \alpha)\ =\ m\ a_Z \tag{5-32}$$

Because the distance between the trunnion and center of gravity of the gun is small compared to the length of the gun, it is assumed that the velocity and acceleration of the mass center is equal to that of the trunnion. This is of course justified for small angular velocities. With this assumption, the expression for the acceleration can be simplified, since the terms representing the Coriolos acceleration and the relative acceleration of the gun with respect to the hull can be neglected. Thus, the accelerations, a_Z and a_X, are the accelerations of the hull in the z and x directions, respectively. However, this distance between the mass center and trunnion will remain in the equations which calculate moments. Therefore, any forces applied to the gun by the trunnion will be included in the dynamics and result in a net torque about the y axis.

From Euler's equations, the equation of motion due to rotation can be determined.

$$I_{yy}\,(\ddot{\Phi} + \ddot{\beta}) \quad + \quad (I_{xx} - I_{zz})\,\omega_z\,\omega_x \quad = \quad M_y \qquad (5\text{-}33)$$

where,

ω_x, ω_z = the angular velocity of the hull about the x and z axis, respectively,

I_{xx}, I_{yy}, I_{zz} = the principal moments of inertia,

M_y = the moment of the total external forces about the y axis.

The moment can be expressed in terms of the forces as follows:

$$M_y \quad = \quad -0.12\ F_z \quad + \quad (38.4)\ f\ \sin\psi \quad - \quad \text{fric}\ (\dot{\Phi}) \qquad (5\text{-}34)$$

where fric is a function of $\dot{\Phi}$ and represents the damping in the trunnion. This function can be expressed as $\gamma\dot{\Phi}$ when modelled as viscous friction (See section 5.3.3.1.) with the coefficient γ having a value of 42,000 in-lb/rad/sec.

Substituting equation (5-34) into (5-33) and modelling the damping in the trunnion as viscous friction, the equation of motion for the system can be rewritten as:

$$I_{yy}\,(\ddot{\Phi} + \ddot{\beta}) \quad = \quad -0.12\ m\ [a_z \ + \ g\ \sin(\Theta + \alpha)] \ + \ (38.28)\ f\ \sin\psi$$
$$- \ (I_{xx} - I_{zz})\omega_x\omega_z \ - \ \gamma\dot{\Phi} \qquad (5\text{-}35)$$

Substituting equation (5-25) into (5-35) and solving for the angular acceleration of the gun with respect to the hull, Φ, we get:

$$\ddot{\Phi} \ = \ \frac{-0.12\ m}{I_{yy}}\,a_z \ + \ \frac{566.54\ \sin\Theta\ f}{I_{yy}\ \ell} \ - \ \frac{0.12\ m\ g\ \sin(\Theta + \alpha)}{I_{yy}}$$
$$- \ \frac{\gamma\ \dot{\Phi}}{I_{yy}} \ - \ \frac{(I_{xx} - I_{zz})\omega_x\omega_z}{I_{yy}} \ - \ \ddot{\beta} \qquad (5\text{-}36)$$

5.3.2.3. Linearizing the gun model. The equations for the gun can be simplified by using the first two terms of the Taylor Series to linearize the model. Φ is a function of six variables: a_z, f, Φ, Φ, ω_x, and ω_z. After taking a first order approximation of the Taylor series about the points 0, f_0, Φ_0, 0, 0, and 0, respectively, and substituting equations (5-23) and (5-27) into equation (5-36) we can write Φ and ℓ as linear differential equations.

$$\ddot{\Phi} \ = \ A\Phi \ + \ BP \ + \ Ca_z \ - \ F\dot{\Phi} \ + \ G \qquad (5\text{-}37)$$
$$\dot{\ell} \ = \ D\dot{\Phi} \qquad (5\text{-}38)$$

where,

P = the pressure which causes a force in the actuator ($P = f/A_p$) where A_p is the area of the piston.

$$A = \frac{0.12\ m\ g \sin(\Phi_0)}{I_{yy}} - \frac{566.54\ f_0 \cos(1.26 - \Phi_0)}{I_{yy}\{\mathrm{sqrt}\,[1684.4 - 1131.1 \cos(1.26 - \Phi_0)]\}} + \frac{(566.54)^2 f_0 \sin^2(1.26 - \Phi_0)}{I_{yy}\{\mathrm{sqrt}\,[1684.4 - 1131.1 \cos(1.26 - \Phi_0)]\}}$$

$$B = \frac{566.54\ A_p \sin(1.26 - \Phi_0)}{I_{yy}\{\mathrm{sqrt}\,[1684.4 - 1133.1 \cos(1.26 - \Phi_0)]\}}$$

$$C = \frac{-0.12\ m}{I_{yy}}$$

$$D = \frac{-566.54 \sin(1.26 - \Phi_0)}{\{\mathrm{sqrt}\,[1684.4 - 1133.1 \cos(1.26 - \Phi_0)]\}}$$

$$F = -\frac{\Upsilon}{I_{yy}}$$

$$G = \frac{-0.12\ m\ g \cos(\Phi_0)}{I_{yy}} - \frac{0.12\ m\ g \sin(\Phi_0)\ \Phi_0}{I_{yy}} + \frac{566.54\ f_0 \cos(1.26 - \Phi_0)\ \Phi_0}{I_{yy}\{\mathrm{sqrt}\,[1684.4 - 1131.1 \cos(1.26 - \Phi_0)]\}} - \frac{(566.54)^2 f_0 \sin^2(1.26 - \Phi_0)\ \Phi_0}{Iyy\{\mathrm{sqrt}\,[1684.4 - 1131.1 \cos(1.26 - \Phi_0)]\}}$$

5.3.2.4. Obtaining steady-state values. The values for the coefficients in expressions (5-37) and (5-38) were obtained by using a small program "eval" which was written to compute the average of steady - state values for a chosen range of Φ. The program asks the user to enter the value (or range) of Φ. It then determined the actuator length, ℓ, using equation (5-23). The steady - state force, f, needed to obtain a gun displacement of Φ was then computed using equation (5-36) with the assumption that $\dot{\Phi}$, a_z, ω_x, ω_z, and $\dot{\beta}$ have a value of zero at steady - state. Since the actuator force, f, is just the pressure times the area of the piston ($f = PA_p$), "eval" actually determines the steady - state pressure necessary to obtain an angle Φ. "eval" then calculates the coefficients by setting the steady - state values for the force equal to f_0, and the user - selected value of Φ equal to Φ_0 in equation (5-37) and (5-38). The coefficients were averaged for a range of Φ_0 between -10 and 45 degrees and found to be:

$A = -4.040 \times 10^{-4}$	$1/s^2$
$B = 1.405 \times 10^{-3}$	$1/(psi\ s^2)$
$C = -4.847 \times 10^{-5}$	$1/in$
$D = 13.631$	in
$F = 0.917$	rad/sec
$G = -1.641 \times 10^{-2}$	$1/s^2$

Thus, the equations become:

$$\ddot{\Phi} = (-4.04 \times 10^{-6})\ \Phi + 0.00145\ P + (-4.88 \times 10^{-5})\ a_Z - 0.917\ \dot{\Phi} - 0.0164$$

$$\dot{\ell} = (13.631)\ \dot{\Phi} \qquad (5\text{-}39,\ 5\text{-}40)$$

5.3.2.5. Transfer function of the gun. The transfer function for the gun can be determined by representing equations (5-37) and (5-38) in the frequency domain, where "s" is the LaPlace operator.

$$\Phi(s) = \frac{B\ s\ P(s)}{s^2 + Fs - A} + \frac{C\ s\ a_Z(s)}{s^2 + Fs - A} + \frac{G\ s}{s^2 + Fs - A} \qquad (5\text{-}41)$$

Using the values for the coefficients as described in the previous section (e.g., in equations (5-39) and (5-40)), the value of A is very small and, more importantly, is much less than the value of F. For this reason, the coefficient A has little effect on the poles of the gun and so will be neglected in the following analysis. With the value of A assumed negligable, it can be easily seen that the poles due to the gun are at s = 0 and -0.917 rad/sec. However, the pole and zero at the origin cancel, leaving a single pole at -0.917.

5.3.3. Modelling the Trunnion Damping.

5.3.3.1. Viscous friction. If we assume the gun is level (e.g., $\Phi = 0$), then the actuator length, ℓ, and the angle between the actuator and the gun, ψ, can be calculated using equations (5-23) and (5-24) from section 5.3.2.1.

$$\ell = \mathrm{sqrt}\ \{ (14.1)^2 + (38.28 - 4.5)^2 \} = 36.6 \text{ in.}$$

Since $(\Theta + \alpha) = 90°$, and α was calculated to be 17.7° in equation (5-26), Θ must have a value of 72.3°. Now, from equation (5-25) we can determine the value of ψ

$$\psi = \arcsin \left[\frac{14.8 \sin\Theta}{\ell} \right] = 22.6°$$

The next step involves determining what force or torque is needed to move the gun from its horizontal position. The pressure necessary to move the gun is about 60 psi. Using this nominal pressure, the force can then be calculated.

$$f = PA = (4.72)(60) = 283.2 \text{ lb.}$$

Using the notation of section 5.3.2., the force can be represented with respect to the "B" coordinate system as follows:

$$f\ \vec{e}_1 = f\ (-\cos\psi\ \vec{i} - \sin\psi\ \vec{k}) = -261.4\ \vec{i} - 108.7\ \vec{k}$$

The necessary torque to move the gun can now be computed using the distance between the point of application of the force and the trunnion, and the cross product.

$$\text{Torque} = \vec{r} \times \vec{f} = 38.4\, \vec{i} \times (-261.4\, \vec{i} - 108.7\, \vec{k}) \approx 4200 \text{ in-lb } \vec{j}$$

We can now use this value to represent the damping in the trunnion as viscous friction. A simple way of doing this is to assume that the friction has a coefficient of 4,200 when the angular velocity of the gun is at half of its maximum value, and a value of -4,200 when the velocity half of its minimum value. The rate sensor which measures the gun elevation rate saturates for inputs greater than 0.175 rad/sec. So if we assume that the angular velocity of the gun, $\dot{\Phi}$, has a magnitude less than 0.2 rad/sec, we can model the damping as viscous friction, $42{,}000\dot{\Phi}$. (See Figure 5-10)

5.3.3.2. Coulomb friction. On the other hand, it is also possible to represent the damping as coulomb friction. To model nonlinear friction, we can assume a steep slope for angular velocities near zero and a constant magnitude opposing the relative gun motion for all other velocities. Using 1/20 of the maximum velocity (0.01) as the bounds on the linear portion, the steep slope is calculated to have a value of 42,000 in-lb/rad/sec for angular velocities of magnitude less than 1/100 rad/sec. For larger magnitudes, the friction may be modelled as a constant of ± 4,200 in-lb, of magnitude opposing the relative motion of the gun. (See Figure 5-11)

5.3.4. Hydraulics of the Gun.

5.3.4.1. Elevation load pressure - fluid flow relationship. The hydraulic system for the gun is modelled and a block diagram is shown in Figure 5-12. The variable Q represents the flow out of the pressure control servo valve, $\dot{\ell}$ is the piston rod velocity, and P represents the load pressure. The other variables involved are defined in Appendix A. From the diagram, it is possible to find the equation which determines P from Q and $\dot{\ell}$.

$$P(s) = \frac{(Q(s) - A_p \dot{\ell})\, 2\beta}{V s + 2\beta K_L} \tag{5-42}$$

5.3.4.2. Pressure control servo valve. A block diagram of the elevation servo valve is in Figure 5-13. The model is identical to that found in the catalog supplied by the manufacturer, MOOG, for the Series 15 Pressure Control Servovalve. Using the nominal parameters given by the manufacturer, the relationship between the output flow of the valve, Q, and the two "inputs" (the input current to the valve, I and the load pressure P which is fed back from the gun itself) can be expressed as:

$$Q(s) = \frac{(K_{Q1} K_{Q2} K_{TM} A_1)\, I(s) - (K_{Q1} \ell_N A_N + K_{PQ1} K_F) K_{Q2} A_2 P(s)}{K_F A_1^2 s + K_B (K_{Q1} A_N \ell_N + K_{PQ1} K_F)} \tag{5-43}$$

5.3.4.3. Combining pressure - fluid flow relationship with servo valve. Block diagrams 5-12 and 5-13 can be combined to determine the total

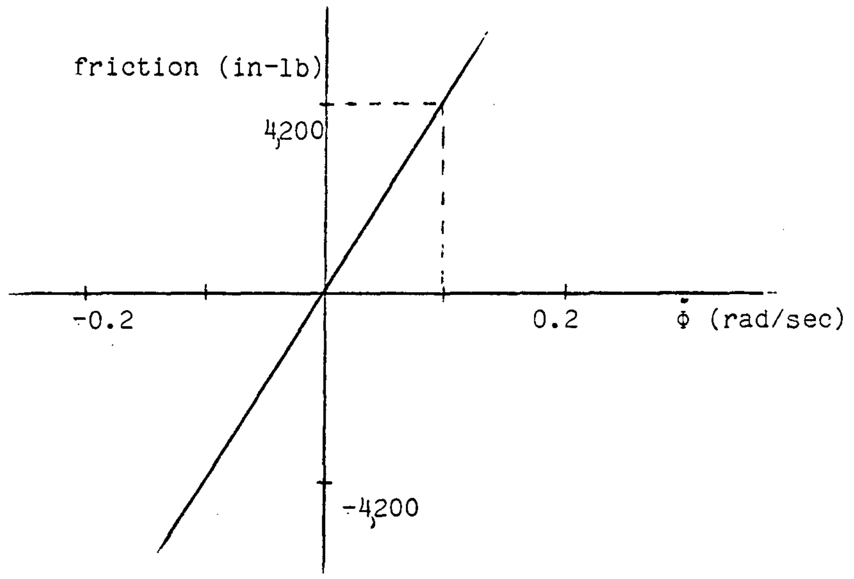


Figure 5-10. Modelling viscous friction

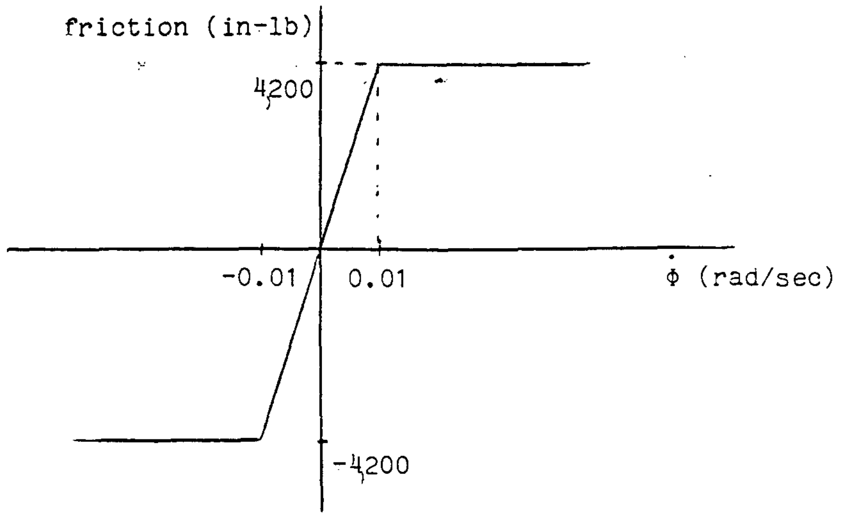


Figure 5-11. Modelling Coulomb friction

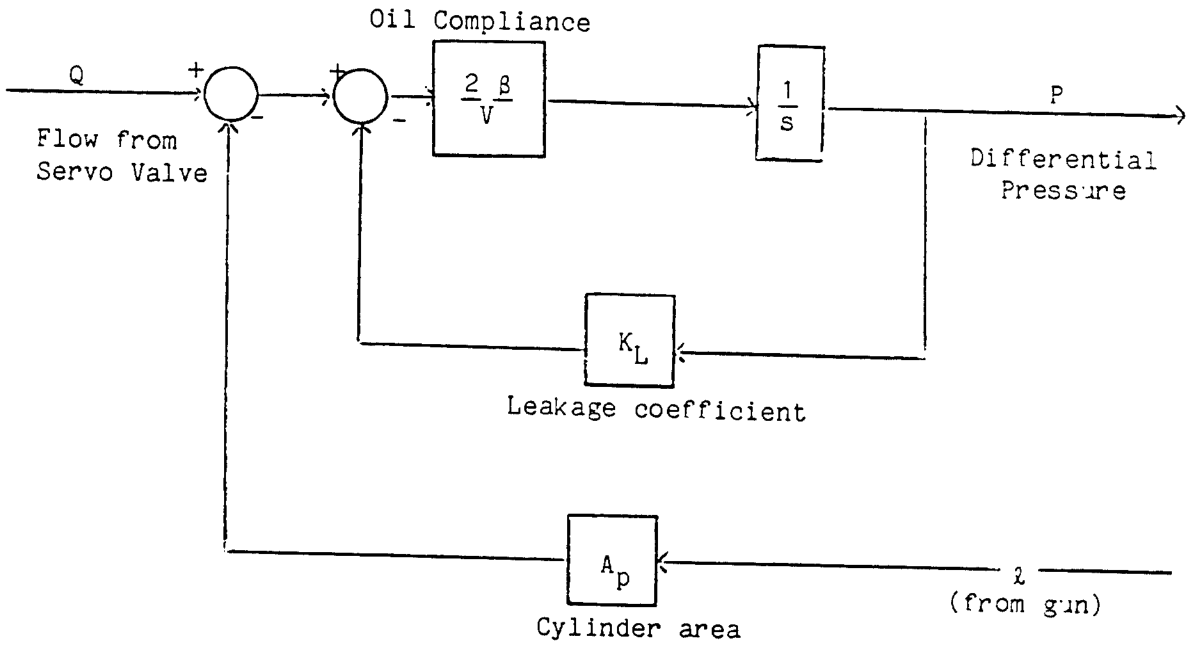


Figure 5-12. Hydraulic System

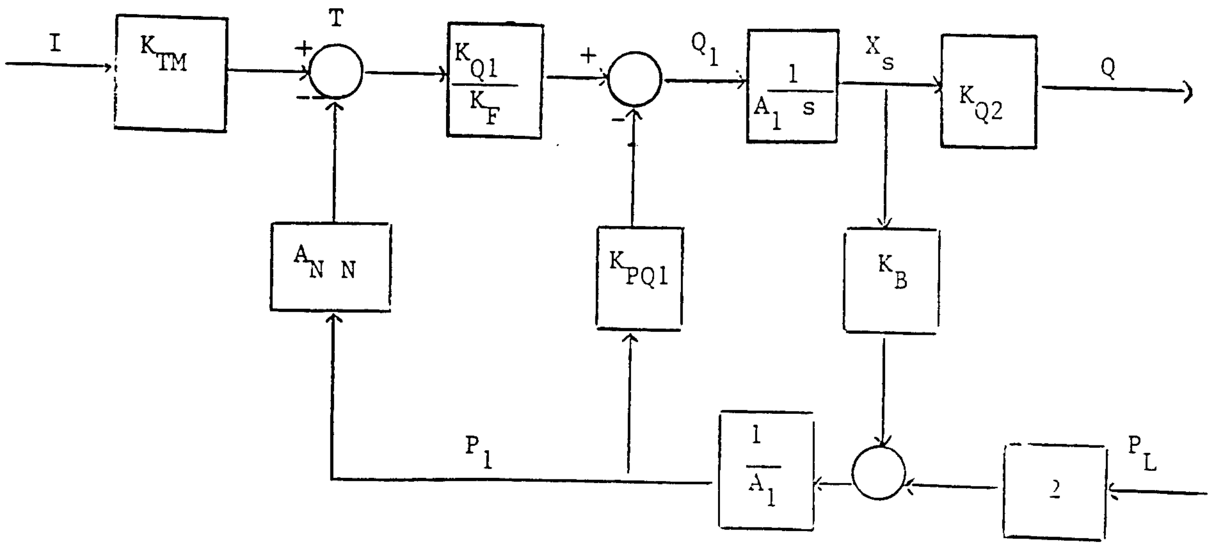


Figure 5-13. Elevation Servo Valve

transfer function for the hydraulics of the system. Therefore, substituting equation (5-43) into (5-42), and solving for the pressure, the relationship between load pressure and input current and actuator velocity can be determined.

$$P(s) = \frac{2\ \beta\ (K_{Q1}\ K_{Q2}\ K_{TM}\ A_1)\ I(s)\ -\ 2\ \beta\ A_D\ (K_F\ A_1^2\ s + KK\ K_B)\ \dot{\ell}}{(K_F\ A_1^2\ s + K_B\ KK)\ V\ s + 2\ \beta\ K_{Q2}\ A_2\ KK + 2\ \beta\ K_L (K_F\ A_1^2\ s + KK\ K_B)} \qquad (5\text{-}44)$$

where $KK = (K_{Q1}\ \ell_N\ A_N + K_{PQ1}\ K_F)$.

Substituting in the values as listed in Appendix A, the transfer function for the hydraulics of the system can be written as:

$$P = \frac{39{,}573{,}170.73\ I\ -\ (36{,}307.69\ s + 6{,}294{,}281.76)\ \dot{\ell}}{s^2 + 181.05\ s + 139{,}777.47} \qquad (5\text{-}45)$$

where,

I = the input current (mA),

$\dot{\ell}$ = the actuator velocity (in/sec).

Thus, the poles of the servo valve are located at $s = -90.53 \pm 362.74\ i$.

5.3.5. Open Loop System.

From the above sections a block diagram of the open loop linearized system can be constructed. (See Figure 5-14). The voltage representing the velocity of the gun can be expressed as:

$$\omega_V = K_T\ (\omega - \dot{\beta}) \qquad (5\text{-}46)$$

where,

K_T = the gain of the rate sensor = 150 volts/rad/sec

$\dot{\beta}$ = the velocity of the hull as defined in section 5.3.2.2.

$$\dot{\Phi} = \omega(s) = \frac{150\ \ (B\ z\ I(s)\ +\ (C\ a_z(s)\ +\ G)(s^2 + K_1\ s + K_2)}{(s + F)(s^2 + K_1\ s + K_2)\ +\ B\ D\ (xs + y)} \qquad (5\text{-}47)$$

where,

s = the LaPlace operator,

B, C, D, F, and G are the averaged parameters of the gun as defined in section 5.3.2.4:

$B = 1.405 \times 10^{-3}$ 1/(psi s^2)

$C = -4.847 \times 10^{-5}$ 1/in

$D = 13.631$ in

$F = 0.917$ rad/sec

$G = -1.641 \times 10^{-2}$ 1/s^2

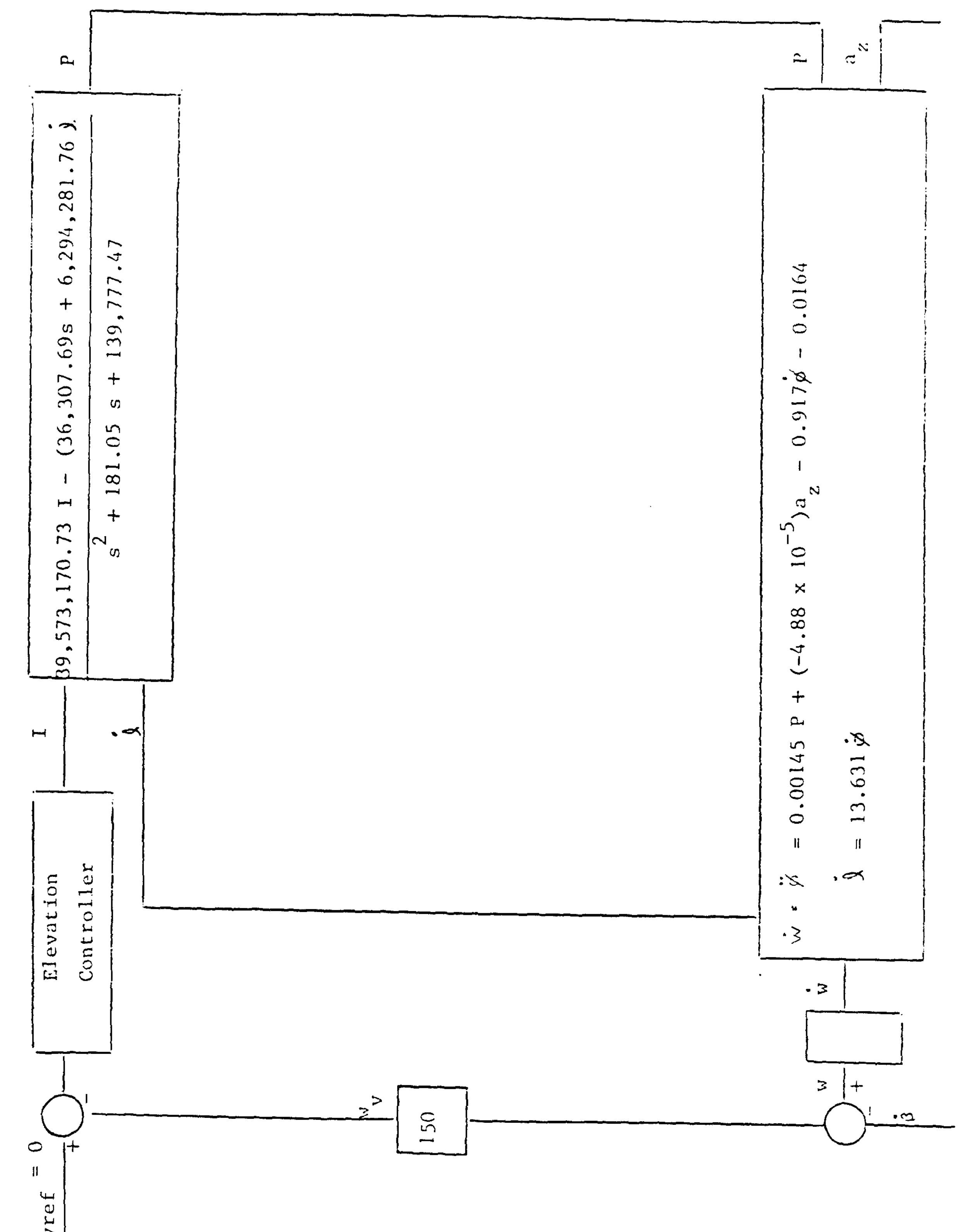

$\omega_{vref} = 0$
Elevation Controller
I
$\dfrac{39{,}573{,}170.73\ I - (36{,}307.69s + 6{,}294{,}281.76)\dot{\lambda}}{s^2 + 181.05\ s + 139{,}777.47}$
P
$\dot{\lambda}$
ω_v
150
$\dot{w} \cdot \ddot{\phi} = 0.00145\ P + (-4.88 \times 10^{-5})a_z - 0.917\dot{\phi} - 0.0164$
$\dot{\lambda} = 13.631\dot{\phi}$
a_z
$\dot{w}$
w
$\dot{\beta}$

Figure 5-14. Block Diagram of Elevation Stabilization System

x, y, z, K_1, K_2 are the elevation servo valve parameters as found in section 5.3.4.3.

$$x = 36{,}307.69 \quad \text{psi/in}$$
$$y = 6{,}294{,}281.76 \quad \text{psi/in sec}$$
$$z = 39{,}573{,}170.73 \quad \text{psi/s}^2 \text{ mA}$$
$$K_1 = 181.05 \quad 1/\text{sec}$$
$$K_2 = 139{,}777.47 \quad 1/\text{sec}^2$$

The program "eval" (See section 5.3.2.4.) not only computes the average parameters for the gun, but also determines the open loop poles for the system. The poles were found to be at $-90.1 \pm 363.6i$ and -1.78. Also, "eval" was used to compute the dc gain of the system, which was 0.22217.

5.3.6. Simplified Models.

5.3.6.1 First order approximation. The above system can be simplified to a first order system by "ignoring" the complex poles ($-90.1 \pm 363.61\, i$). Keeping the dc gain constant, the first order system can be represented by:

$$\omega(s) = \frac{0.395 \; I(s)}{s + 1.78} \qquad (5\text{-}48)$$

This model has a pole at -1.78 as desired and a dc gain equal to that of the orginal third order system. The input to the system is the current into the valve. The constant input denoted by G in the above sections has a value of only 1.714×10^{-2} and so it shall be neglected. Also, the coefficient of the input a_z is only 5.410×10^{-3}. Unless the acceleration is very large, which is impractical, this factor will also be small compared to the input due to the current. For instance, even if the tank is accerating at 1 g (32.2 ft/sec) and we assume that this acceleration is completely in the "z" direction (this is not necessarily the vertical direction since "z" is a body fixed axis in the gun itself), the term due to the input a_z would only have a value of about 2.1 sec^{-1}. Thus, this input is also neglected.

This is the system that was used to develop a control law using the program "DTQD" (See section 5.2.) It may be expressed in state-space notation as:

$$\ddot{\Phi} = -1.78\,\dot{\Phi} + 0.395\, I \qquad (5\text{-}49)$$

5.3.6.2. First order system with Coulomb friction. In the previous models the damping in the trunnion has been treated as viscous friction. In particular, the modelled friction has had a coefficient of ($42{,}000/I_{yy}$) in-lb/rad/sec. Friction is the primary parameter which causes the dominant pole to lie at -1.78.

A second representation of the first order system can be made by modelling the damping in the trunnion as nonlinear friction. In section 5.3.3.2., friction was modelled as nonlinear coulomb friction with a magnitude of 4,200/Iyy in-lb for gun velocities greater than 0.01 rad/sec. For magnitudes less than 0.01 rad/sec, friction was essentially modelled as viscous friction with a very large coefficient. In fact, using this model for friction, the entire system can be described by:

$$\ddot{\Phi} = \begin{cases} 0.395\,I - 0.0917; & \text{for } \dot{\Phi} > 0.01 \\ 0.395\,I + 0.0917; & \text{for } \dot{\Phi} < -0.01 \\ -9.17\,\dot{\Phi} + 0.395\,I; & \text{for } -0.01 < \dot{\Phi} < 0.01 \end{cases} \qquad (5\text{-}50)$$

5.3.6.3. Third order system. Another model of the system used to "check" the control law was the model of the complete third order system. This model is very similar to that of equation (5-46) except the velocity of the hull, $\dot{\beta}$, is omitted and only a single input, the current into the valve, is modelled. Systems implementing β are considered in section 5.3.6.4. The other inputs, denoted by G and a_z above, are ommitted for the reasons explained in 5.3.6.1. The model can be represented by:

$$\omega(s) = \frac{55493.2 \quad I(s)}{[s + (90.1 \pm 363.6i)][s + 1.78]} \qquad (5\text{-}51)$$

Using ω, ω, and ω as states, the state - space representation of the system is :

$$\underline{\omega} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -249778.14 & -140645.60 & -181.98 \end{bmatrix} \underline{\omega} + \begin{bmatrix} 0 \\ 0 \\ 55493.2 \end{bmatrix} I \qquad (5\text{-}52)$$

5.3.6.4. Models with a "Disturbance". In all the simplified models derived thus far, the term β, the angular velocity of the hull of the tank in the "y" direction, has been neglected. This term may be added to any of the above three models to create three more representations of the system. β is simply treated as an uncontrolled input (i.e., a disturbance) into the system, as seen in Figure 5-14. In our case, we set β equal to a sinusoidal waveform whose amplitude and frequency could be arbitrarily chosen.

5.3.7. Using "DTQD".

To obtain a control law for the original simplified model - the first order system with linear friction and no "disturbances," the program "DTQD" was implemented. To use "DTQD," we first had to determine various parameters of the system including the sampling time, τ, and the number of quantization steps.

Keeping in mind that our actual system did have a "disturbance" which caused the occurance of $\dot{\beta}$, the angular velocity of the hull, we determined the appropriate value for τ. It was assumed that the highest frequency for the hull velocity is approximately 15 Hz. The sampling rate was chosen such that it would be possible to sample about six times per cycle. Therefore, τ = 0.01 was picked.

Thus, the discrete-time representation of this system is:

$$x(K+1) = 0.982\,x(K) + 0.00392\,I(K) \qquad (5\text{-}53)$$

To determine the number of quantization steps, we first had to know the upper and lower limits for our state and input. Because the rate sensor which measures the gun elevation rate saturates for inputs greater than 0.175 rad/sec, we chose ± 0.2 rad/sec as our upper and lower bound for the state. The input bounds were given by the servo valve manufacturer to be ± 10 mA.

In choosing the number of quantization steps, the following "rule" was implemented: Use twice the number of steps which takes the state from its maximum value down to 10% of its upper bound.

$$0.2\,e^{(-1.78)(0.01)x} = 0.02 \quad ==> \quad x \approx 129$$

Thus, the desired number of quantization steps was chosen to be 256.

In quantizing the input, it was desired that a change of one step in the input would approximately cause a change in one step of the state. Thus, the following ratio was desired:

$$\frac{0.2}{256} = \frac{10\ (0.00392)}{x}$$

where 0.2 and 10 represent the upper bounds of the state and input, respectively; 256 and x represent the number of quatization steps for the state and input, respectively; and 0.00392 is the input "matrix" for the corresponding discrete-time model. Solving the ratio for x we find that the number of input steps is approximately 32.

5.4.0. Simulation Results

5.4.1. The Model.

The previous section developed a mathematical model for the elevation actuator of the M60 tank. After the system was linearized, the actuator was a third order system with a pole at -1.78, and two complex poles at -90±360j. If the two complex poles are ignored, the resulting model of the elevation dynamics becomes a simple first order system with a pole at -1.78. This simple first order system was used to derive the control law for the elevation system. After the controller was developed, it was simulated using more accurate models of the elevation dynamics. The resulting response should be a reasonable approximation of the actual response expected if the controller was used on the vehicle. As might be expected, there were significant deviations from the response of the idealized first order system. However, with careful selection of the controller parameters some of these problems can be minimized. These relations are explored in the text that follows.

5.4.2. The Control Law.

A control law can be constructed using the theory presented in section 1.0 for the system

$$dx/dt = -1.78\, x(t) + 0.395\, u(t) \qquad (5\text{-}54)$$

The particular gyro used on the the elevation controller saturates at V_{out} = ±0.2 volts. So these were used as the limits on the state for building the control law. Likewise the hydraulic servo valve saturates at I_{in} = ± 10 ma. Thus this was used to define the limits on the input to the system. Considering the expected range of frequencies of the disturbances to the vehicle, a sample time of T = 0.01 sec, or 100 Hz was used. Finally, the state was divided into 256 levels and the inputs into 32 levels. Although these parameters could be changed the resulting system response seems to be well controlled.

Using the system (5-54) and the parameters presented above, a control law based on DTQD system theory can be derived using the program described in 2.0. For this example a minimum time strategy was adopted. Table 5-1 summarizes the results of the control law. The bang-bang characteristiic is quite evident.

Table 5-1. Control Law

Quantized State	Discrete State (volts)	Quantized Input	Discrete Input (mA)
1 to 103	-.198 to -.039	32	10.00
104	-.038	31	9.38
105	-.036	30	8.75
106	-.034	31	9.38
107	-.033	29	8.12
108	-.031	29	8.12
109	-.030	28	7.50
110	-.028	27	6.88
111	-.027	28	7.50
112	-.025	26	6.25
113	-.023	25	5.62
114	-.022	26	6.25
115	-.020	24	5.00
116	-.019	24	5.00
117	-.017	23	4.38
118	-.016	22	3.75
119	-.014	23	4.38
120	-.013	21	3.12
121	-.011	20	2.50
122	-.009	21	3.12
123	-.008	19	1.88
124	-.006	19	1.88
125	-.005	18	1.25
126	-.003	17	0.62
127	-.002	18	1.25
128	0.0	16	0.0
129	.002	15	-0.62
130	.003	16	0.0
131	.005	14	-1.25
132	.006	14	-1.25
133	.008	13	-1.88
134	.009	12	-2.50
135	.011	13	-1.88
136	.013	11	-3.12
137	.014	10	-3.75
138	.016	11	-3.12
139	.017	9	-4.38
140	.019	9	-4.38
141	.020	8	-5.00
142	.022	7	-5.62
143	.023	8	-5.00
144	.025	6	-6.25

145	.027	5	-6.88
146	.028	6	-6.25
147	.030	4	-7.50
148	.031	3	-8.12
149	.033	3	-8.12
150	.034	2	-8.75
151 to 256	.038 to .200	1	-9.38

5.4.3. Step Response.

The closed-loop system driven by the control law presented in table 5-1 was simulated. This section reports some of the characteristics that were observed in the simulations. The portion is divided into several sections. The first section looks into how well the control law acts on the first order system it was designed to control. As expected it performs quite well. The next section examines how well the controller works on the more realistic third order model of the elevation dynamics. Finally, the changes in the response when coulomb friction replaces the linear viscous friction term are examined.

5.4.3.1. First Order Model.

Figure 5-15 shows the response of the system being regulated by the DTQD controller. The system is perturbed by an initial condition of 0.17 rad/sec and the resulting response is plotted. In the early part of the response, a clear minimum time trajectory is shown. (The system is being driven to zero velocity at its maximum acceleration.) After the initial phase of the response, a limit cycle is evident in the output of the system. This is an expected result considering the size of the quantization levels that were used.

Recall from the theoretical development of DTQD system theory, that the grid embedding process allowed large quantization levels without sacrificing accuracy. Figure 5-16 shows how the embedding process improved the response of this first order system. In this example embedding takes place whenever the state is within ±0.02 rad/sec from the origin (the inner 10% of the state space). When this occurs the state and input are scaled by a factor of ten. The effect of the embedding process on the response of the system is clear from the figure. As the system approaches the origin from 0.17 the response is indentical to that of a system without embedding (Fig. 5-15) until the state reaches 0.02. At this point the embedding takes place and the system is slowed. However, since the quantization levels are cut by ten, the system is under the influence of a much more accurate control law, and therefore, the limit cycle behavior is eliminated. As might be expected, adding more embedding cycles does not improve the system response, see Fig 5-17. Therefore, it can be concluded that configuring this system with an embedding process with one or two embedding levels is the best solution for the controller in this situation.

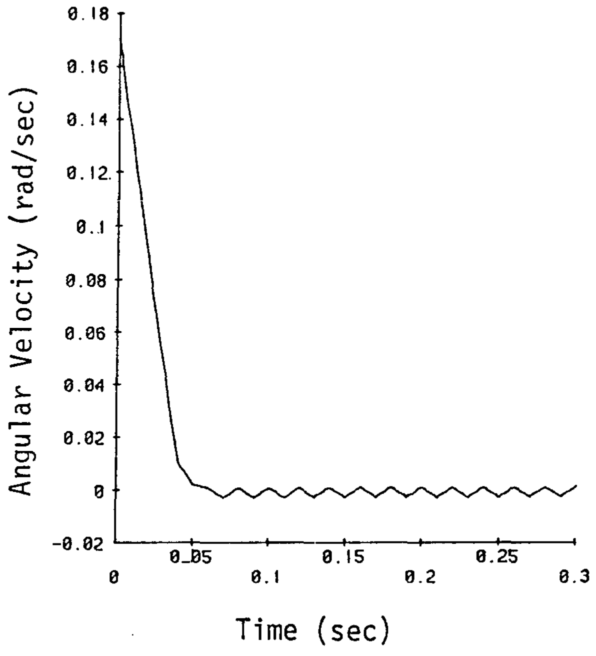


Figure 5-15. First Order System with Zero Embedding Processes

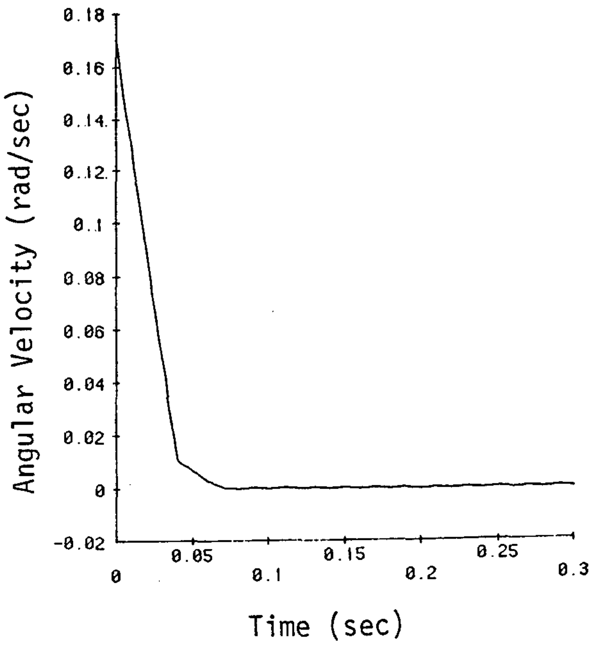


Figure 5-16. First Order System with One Embedding Processes

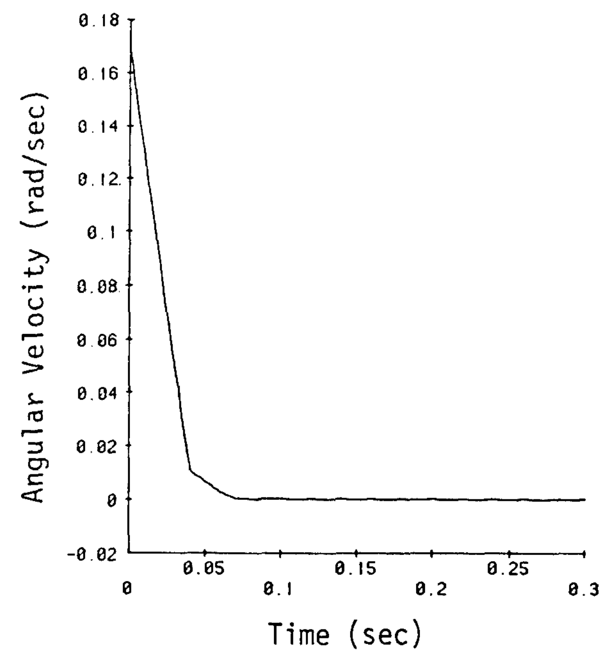


Figure 5-17. First Order System with Two Embedding Processes

5.4.3.2. Third Order System.

The response of the third order model of the gun elevation dynamics for the control law developed in section 5.4.2 is presented in Fig. 5-18 thru Fig 5-21. Again the minimum time response is evident in the graphs. However, the magnitude of the limit cycle has dramatically increased. This is expected since the other two poles are due to a combination of the lag in the valve and the compressibility of the hydraulic fluid. Once again the simulations show that the embedding process will eliminate most of the undesirable characteristics of the response. But, in this case it is advisable to have about 3 to 4 embeddings to damp out all of the limit cycle behavior.

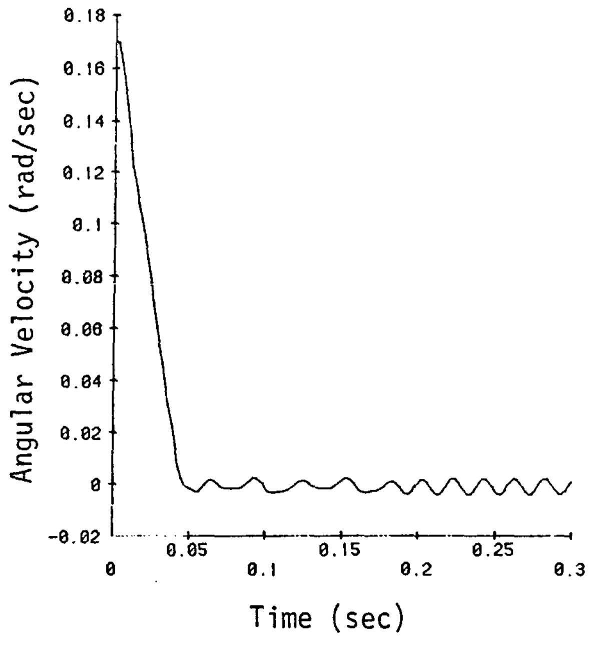


Figure 5-18. Third Order System with Zero Embedding Processes

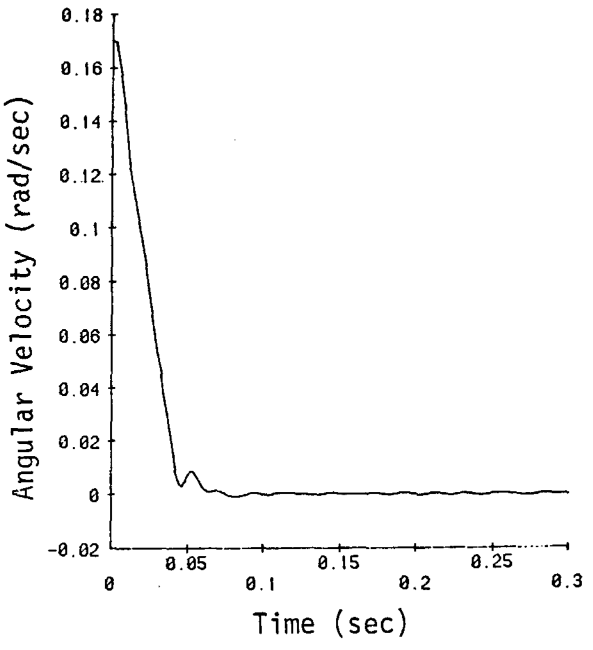


Figure 5-19. Third Order System with One Embedding Processes

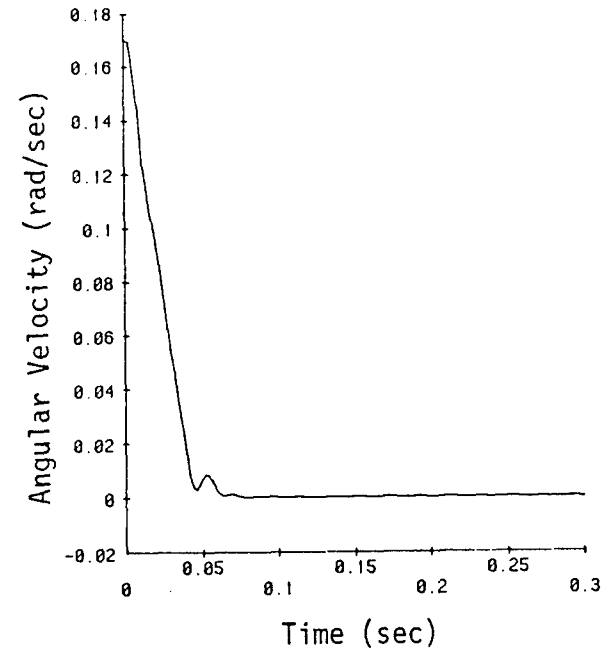


Figure 5-20. Third Order System with Two Embedding Processes

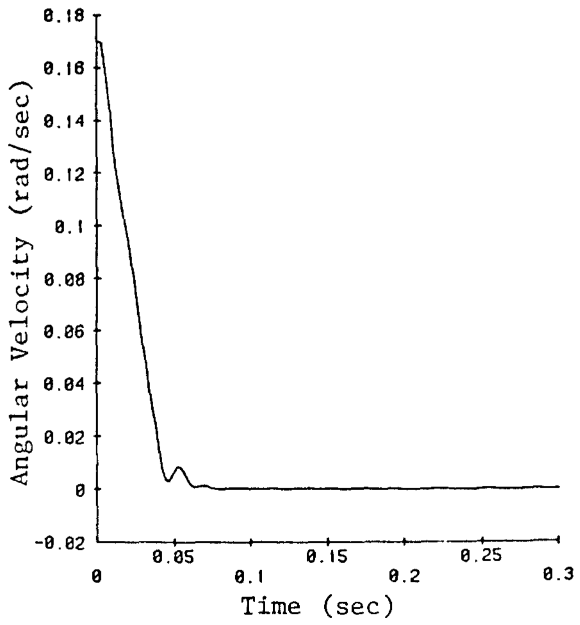


Figure 5-21. Third Order System with Three Embedding Processes

5.4.3.3. Coulomb Friction.

Finally, Fig. 5-22 thru Fig. 5-24 illustrates the response of the first order system with the trunnion modelled with coulomb friction instead of viscous friction. It does not make a large difference whether the friction is modelled with either coulomb or viscous characteristics. This is due to the extremely large gain of the system. The same conclusions can be drawn as for these cases as with the first order system with linear friction. Because of the large quantization levels, a limit cycle will exist unless embedding is used. Two or three embedding levels should be adequate to control the system.

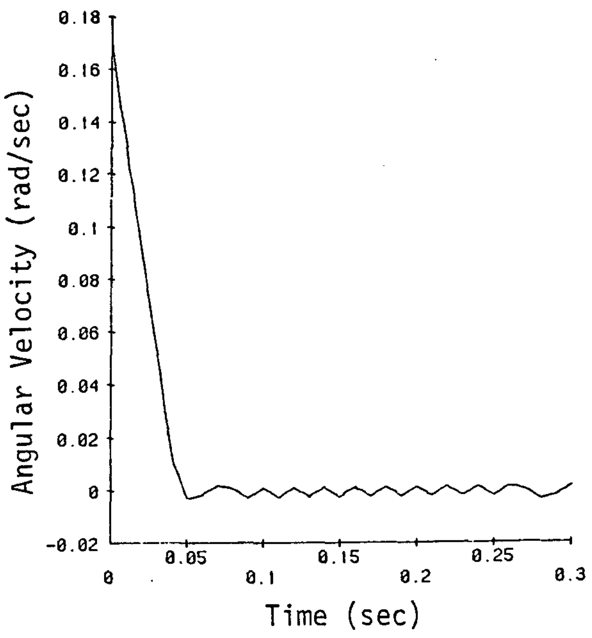


Figure 5-22. First Order System with Coulomb Friction and Zero Embeddings

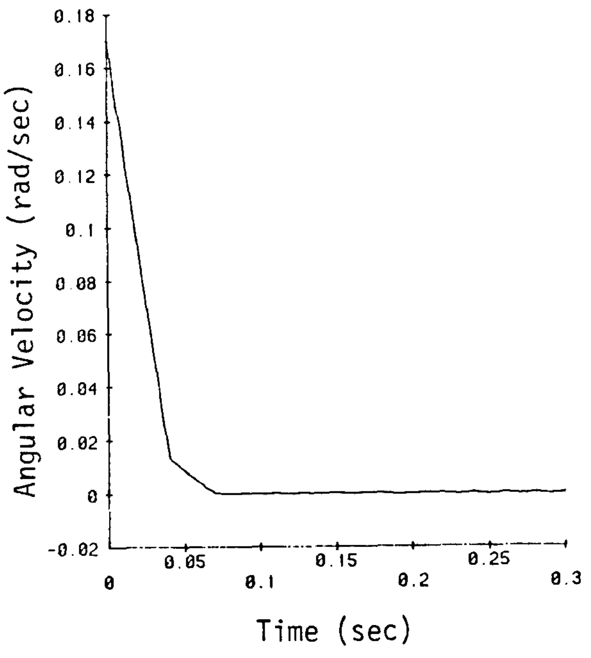


Figure 5-23. First Order System with Coulomb Friction and One Embeddings

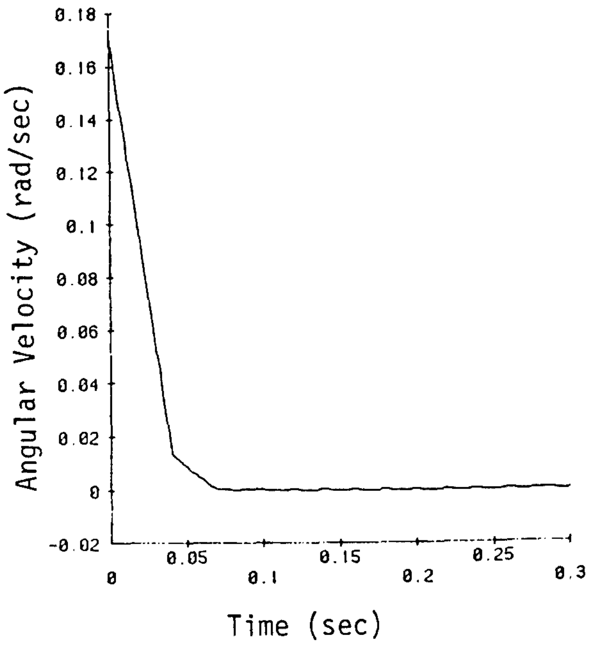


Figure 5-24. First Order System with Coulomb Friction and Two Embeddings

5.4.4. Disturbance Rejection.

The disturbance rejection of this controller will be examined as a final exercise to evaluate the performance of the DTQD controller. In this case the disturbance will be considered to be the velocity of the hull. Although the controller was not specifically designed to reject distubances, it is an interesting excercise to examine its performance in this capacity. Unfortunately, it did not perfom as well in this area as it did as a regulator. This problem will be examined more closely in the follow-on project. Figure 5-25 illustrates the model used to check the rejection capabilities of the controller. As with the step response, we will examine both the first and third order models.

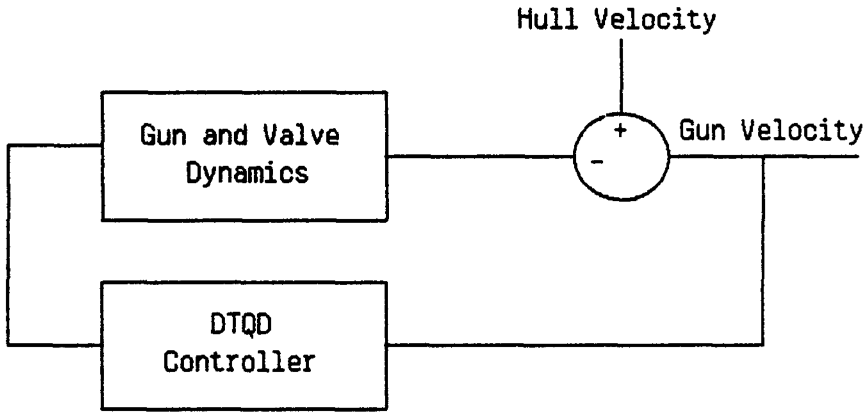


Figure 5-25. Disturbance Model

5.4.4.1. First Order Model.

Figures 5-26 through 5-29 shows a typical response of the first order system with a sinusoidal disturbance, of different frequencies, amplitudes and embedding levels. The amplitudes of the disturbance are attenuated by the controller. Figure 5-30 is a plot of the frequency response of the system due to a sinusoidal disturbance input, without embedding being used. As expected, the lower frequencies (0-5 Hz) are attenuated more than the higher ones (greater than 5 Hz). A second plot of the distubance cancelling effects as a function of frequency of this system is given in Fig. 5-31, however, in this system the embedding process was engaged. The rejection of the sinusoidal inputs for the system with or without embedding are comparable. However, the actual time domain response of the system (Fig. 5-26 thru Fig 5-31) is considerably smoother for the system with embedding. Therefore, embedding improves the rejection capabilities of a system controlled by a DTQD regulator. Only the regulation properties of DTQD controllers have been fully developed. Therefore, it is not surprizing to see the poor rejection responses below. However, the follow on project will look into disturbance rejection extensively.

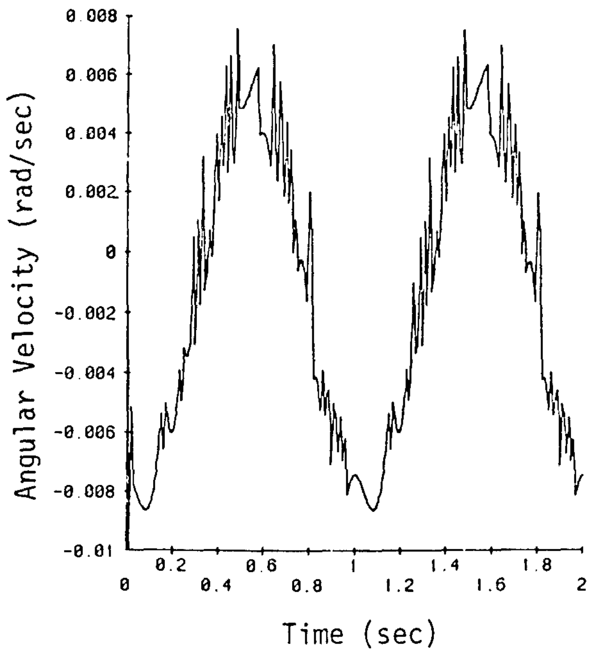


Figure 5-26. First Order System with Zero Embeddings and $d(t) = 0.1 \sin(2\pi t)$

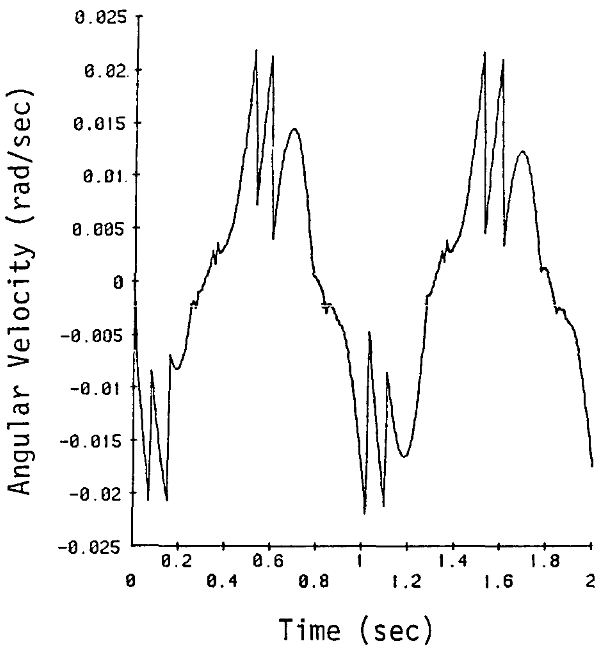


Figure 5-27. First Order System with Three Embeddings and d(t) = 0.1 sin (2πt)

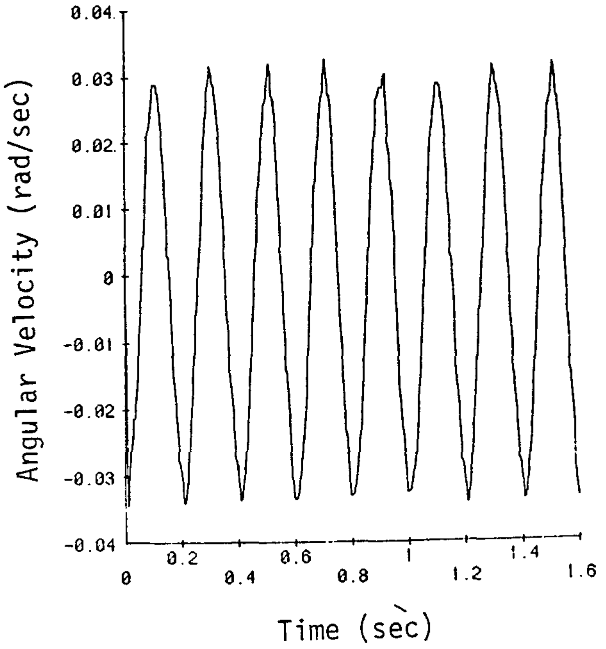


Figure 5-28. First Order System with Zero Embeddings and d(t) = 0.1 sin (10πt)

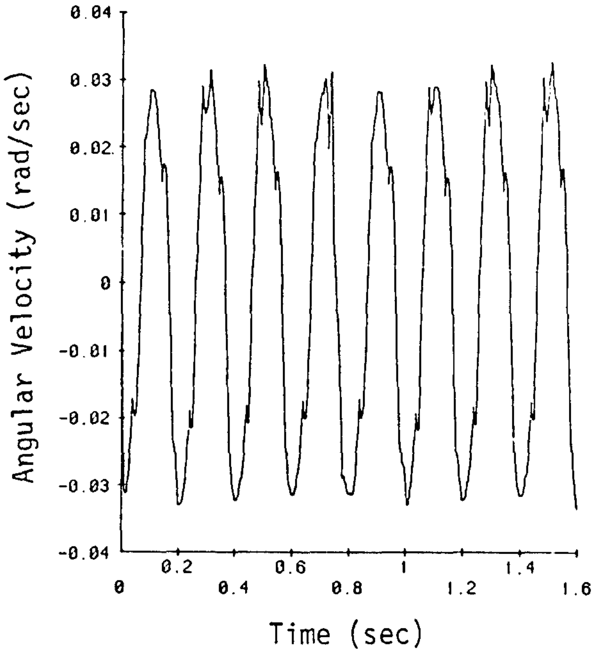


Figure 5-29. First Order System with Three Embeddings and d(t) = 0.1 sin (10πt)

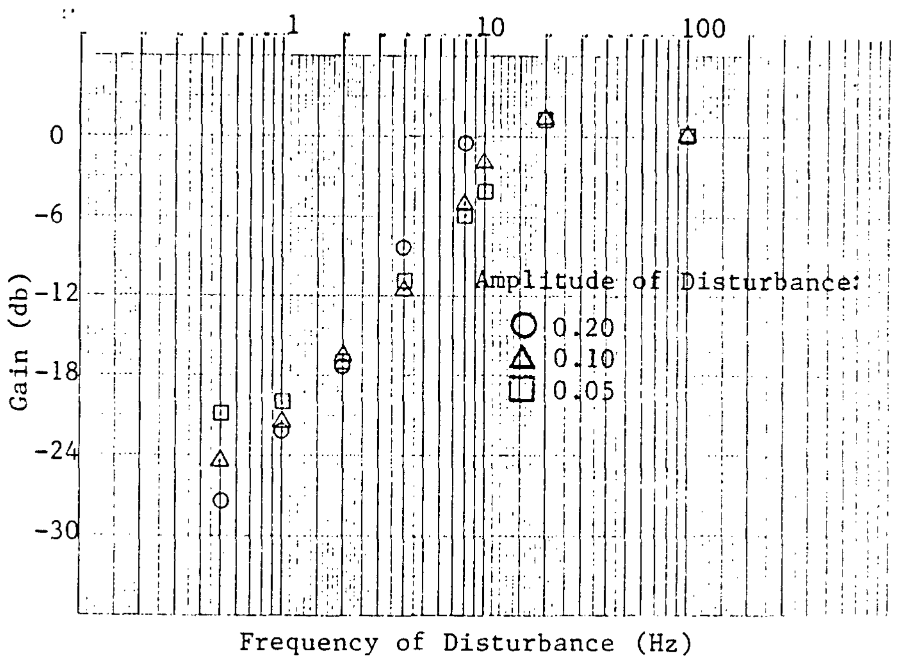


Figure 5-30. Disturbance Frequency Response of the First Order System without Embedding

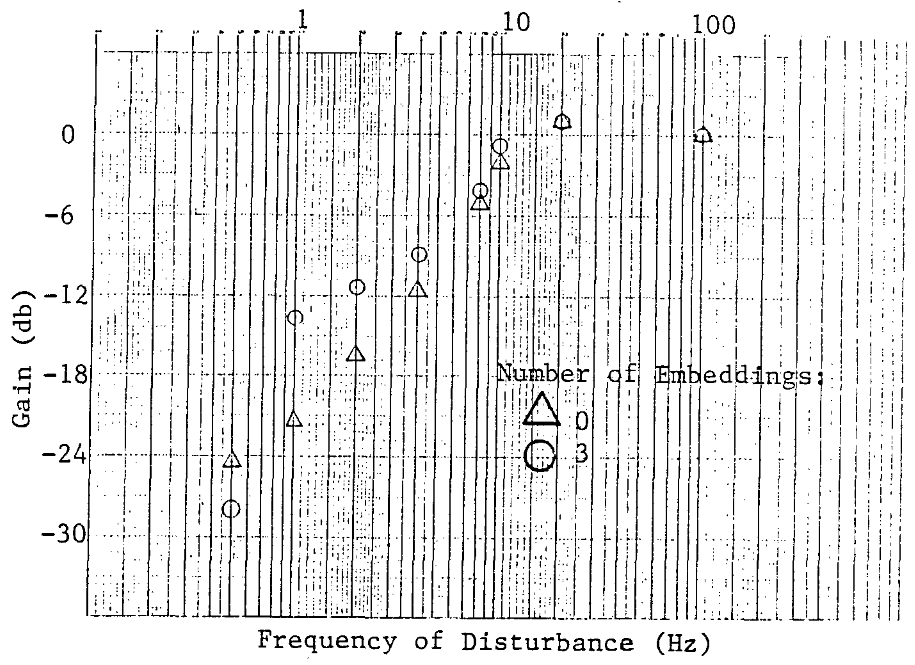


Figure 5-31. Disturbance Frequency Plot of the First Order System with embedding

5.4.4.2. Third Order System.

Figure 5-32 thru Fig. 5-35 are plots of the response of the third order model of the elevation dynamics to a sinusoidal disturbance. The limit cycle behavior of the system is greatly reduced by inceasing the number of embedding levels. As with the step response, about three embedding levels are needed to adequately damp out the high frequency oscillation. A plot of the distubance cancelling effects as a function of frequency of this system is given in Fig 5-36. Remember that at this point in time that only the regulation properties of DTQD controllers have been fully developed. Therefore, it is not surprizing to see the poor rejection responses below. However, the follow-on project will look into disturbance rejection extensively.

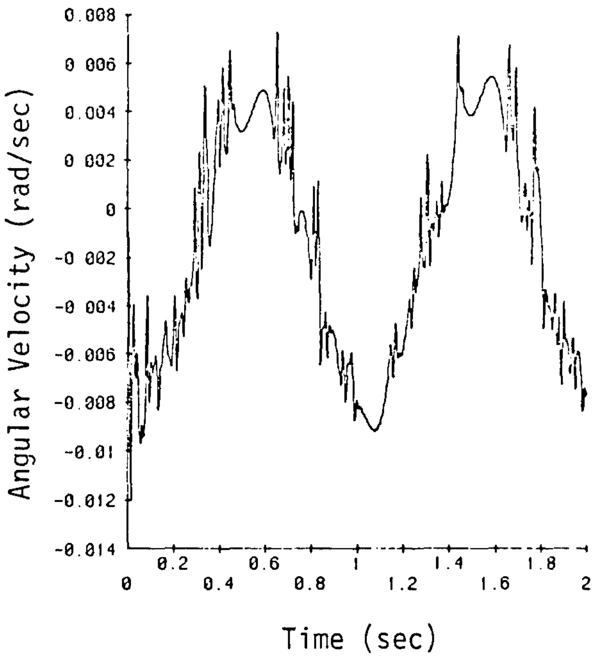


Figure 5-32. Third Order System with Zero Embeddings and d(t) = 0.1 sin (2πt)

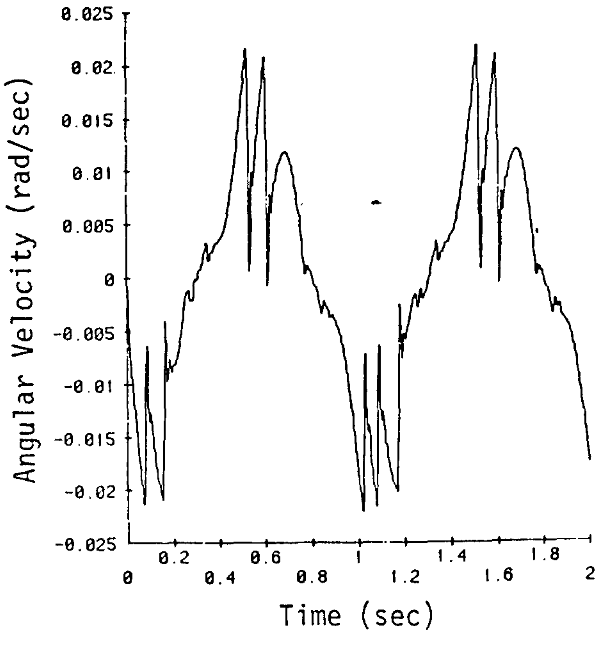


Figure 5-33. Third Order System with Three Embeddings and d(t) = 0.1 sin (2πt)

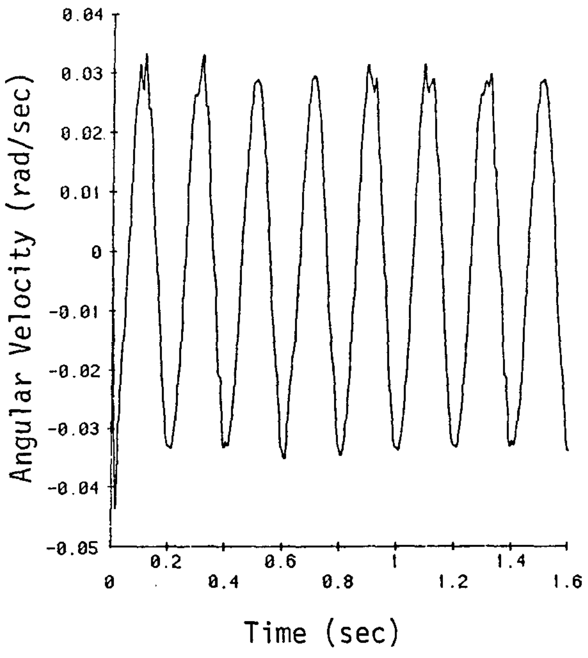


Figure 5-34. Third Order System with Zero Embeddings and d(t) = 0.1 sin (10πt)

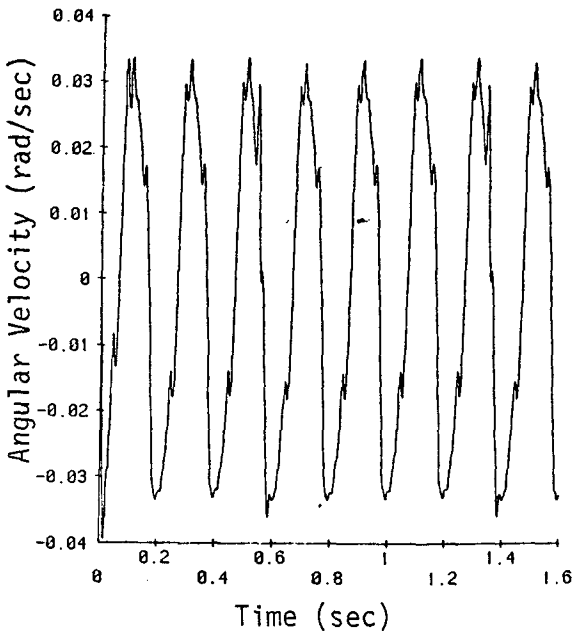


Figure 5-35. Third Order System with Three Embeddings and d(t) = 0.1 sin (10πt)

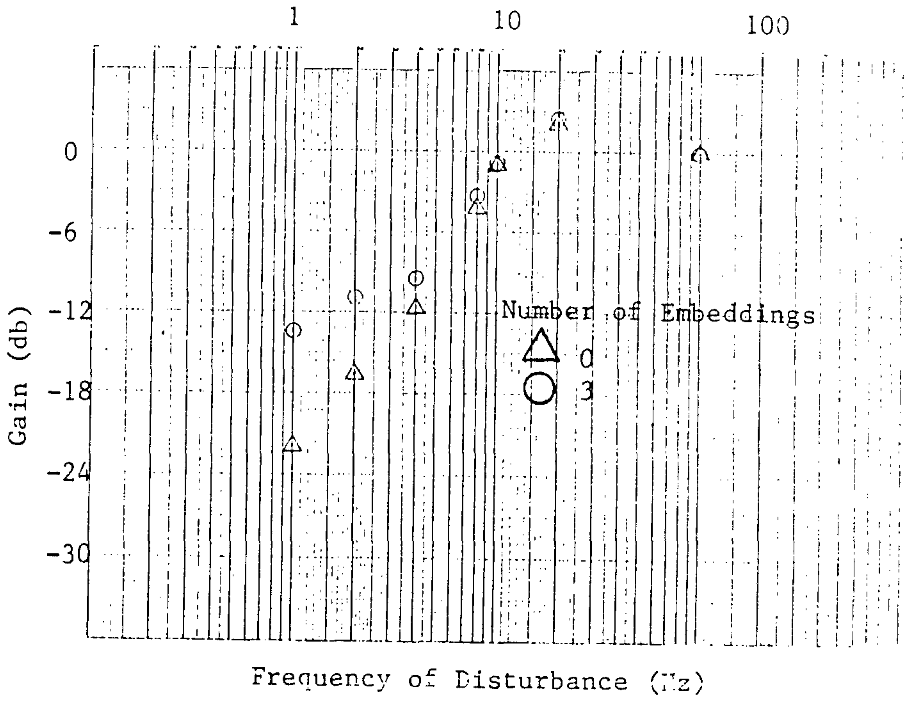


Figure 5-36. Disturbance Frequency Plot of Third Order System without embedding

Compare Fig. 5-33 with Fig. 5-37. The amplitude of the high frequency oscillation is greatly reduced in 5-37, although the same sine wave is forcing the two systems. The reason for this is the embedding process takes place at different times for the two systems. In Fig 5-33 embedding takes place when the system is within the center 10% (.02 rad/sec) of the state space. Fig 5-37, it takes place in the inner 20% (.04 rad/sec). The steady state response of the system in Fig 5-33 is little greater than 0.02 rad/sec. This means the system is constantly jumping from one embedding level to another as the response passes through 0.02 rad/sec. This explains the somewhat erratic behavior of the response in Fig 5-33. With a larger embedding region the steady state response does not cross the the boundary for embedding, and therefore, the response for this system (Fig 5-37) is much smoother.

To obtain the smoother response for all amplitudes of disturbances it is necessary to insure that the response never crosses a boundary for embedding. This is impossible if embedding is done at discrete intervals, since a disturbance can always be found that will have an amplitude which will cross the boundary. For example, if the controller was programmed to embed whenever the state was in the center 20 cells, then a disturbance can be found that would continuously enter and leave the center 20 cells. A method to correct this problem is to use a continuous embedding system. This technique would measure the distance that the state is from the origin and then scale the the states and the inputs by an amount proportional to this distance. Thus, the quantized states would always appeared to the controller to be at approximately the same position from the origin. Also, since embedding takes place continuously, there will be no discrete boundaries will have been shown to add noise to the response. Therefore, it is suggested that this technique be explored in a follow-on project.

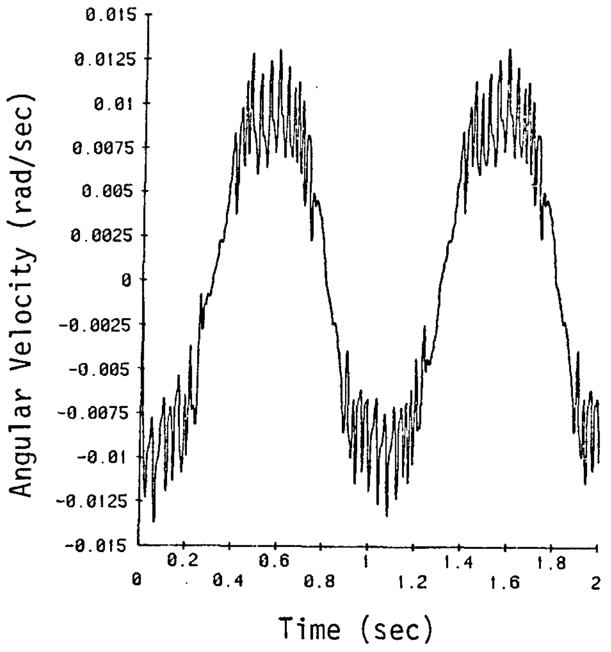


Figure 5-37 Response of the System with Scaling Factor Increased to 20%

THIS PAGE LEFT BLANK INTENTIONALLY

LIST OF REFERENCES

1 Falkenburg, D.R. and Judd, R.P., "A New Approach to Digital Optimal Control of Linear Systems," Proceedings of the 1980 IEEE Midwest Symposium of Circuits and Systems.

2 Falkenburg, D.R. and Judd R.P., "Optimal Control of Discrete Time Quantized Data Systems with Inaccessible States," Proceedings from the Eleventh Pittsburg Conference on Modelling and Simulation.

3 Judd, R.P., Analysis and Control of Discrete Time Systems with Quantized States, Ph.D. Dissertation, Oakland University, 1981.

4 Kalmay, R.E., SM Thesis, MIT Department of Electrical Engineering, 1956.

5 Weng, P.K.C., "A Method for Approximating Dynamical Processes by Finite State Systems", Int. J. Control, Vol. 8, No. 3, pp.285-296, (1968).

6 Kornoushenko, E.K., "Finite Automation Approximation to the Behavior of Continuous Plants," Automatika Telemakhanika, 12, pp.150-157, (1975).

7 Primm R., "Shortest Connection Networks and Some Generalizations," Bell System Tech. J., pp.1389-1401, V. 36, (1957).

8 Nijenhais, A. and Wilif, H.S., Combinatiorial Algorithms, Academic Press, pp. 283-287, (1975).

THIS PAGE LEFT BLANK INTENTIONALLY

APPENDIX A

PARAMETERS VALUES

THIS PAGE LEFT BLANK INTENTIONALLY

1.0. GUN PARAMETERS

M	= mass = 18.5 lb s^2/in
W	= weight = 7141 lb.
I_{xx}	= moment of inertia about x axis = 100 in-lb-s^2
I_{yy}	= moment of inertia about y axis = 45800 in-lb-s^2
I_{zz}	= moment of inertia about z axis = 45800 in-lb-s^2

See also Figure 5-8 in section 5.3.2.

2.0. HYDRAULIC CYLINDER PARAMETERS

β	= Oil compliance = 200,000 lb/s^2
V	= Volume of Hydraulic system = 52 in^3
A_p	= Cylinder area = 4.72 in^2
K_L	= Leakage factor = 0.001

3.0. ELEVATION SERVO VALVE PARAMETERS

I	= Input current = ± 10 mA rated
T	= Torque on armature flapper = ± 0.165 in-lb rated
Q_1	= Hydraulic amplifier flow to drive the spool = ± 0.23 cis max
Q_2	= Servo valve flow, no load = ± 55 cis rated
X_S	= Spool displacement = ± 0.020 in rated
P_1	= Hydraulic amplifier differential pressure = ± 890 psi rated
P	= Load differential pressure = ± 3000 psi rated
K_{TM}	= Torque motor gain = 0.0165 in-lb/ma
K_{Q1}	= Hydraulic amplifier motor gain = 65 cis/radian
K_{Q2}	= Spool flow gain, no load = 8850 cis/in
K_{PQ1}	= Hydraulic amplifier loading effect = 1.26×10^{-4} cis/psi
K_B	= Spool Bernoulli force gradient, no load = 1040
K_F	= Net stiffness of armature/flapper = 45 in-lb/rad
A_1	= Spool driving area = 0.041 in^2
A_2	= Spool feedback end area = 0.0122 in^2
A_N	= Nozzle frontal area = 3.14×10^{-4} in^2
ℓ_N	= Moment arm to nozzles = 0.34 in

THIS PAGE LEFT BLANK INTENTIONALLY

APPENDIX B

PROGRAM DOCUMENTATION FOR "DTQD"

THIS PAGE LEFT BLANK INTENTIONALLY

1.0. INTRODUCTION

The program, "DTQD," aids the user in designing a controller for a discrete time quantized data system. The user enters information regarding the system and data converters, and the program creates the DTQD model of the system. If the quantization levels lead to an acceptible model of the system, the user may then have "DTQD" develop a control law for the system using any desired cost function. Finally, the program lets the user simulate the controlled system, and plot any combination of states, inputs and time.

The program is being developed. Although each stage works, modifications are still being made to make it simpler.

The program is coded in PL1. Although it is currently being run on the Honeywell 68-DPS-2 MULTICS system computer, with slight modifications it could easily be implemented on most main frames. The program consists of the main procedure, DTQD, and 11 external subprograms which are called by DTQD. Each of these procedures may call internal subroutines as well.

2.0. EXTERNAL VARIABLES

The following is an alphabetical list of the external variables used in the program.

a_matrix - A (n x n) array and is the system matrix. (input by user)

b_matrix - A (n x p) array, the input matrix for the system. (input by user)

control_law_file_ptr - A pointer to the beginning of the control law file. (corresponds to the based variable control_law)

cost_function_code - An integer code representing which cost function is to be used. (input by user)

flag.own_quant_file_exists - A one bit variable which designates whether or not a file containing the quantized model exists.

input_cost_matrix - This (p x 1) array is the diagonal elements of the input weighting matrix (i.e. the "R" matrix), which is assumed to be diagonal. It is entered by the user if a quadratic or minimum-control-effort cost function is desired.

job_name - A user-inputted variable representing the name of the current job. It must be one word and contain less than 50 characters. These characters may be any combination of letters, numbers, and underscores; however, the first character must be a letter.

lambda_matrix - A (n x p) array, the discrete - time input matrix for the system. (may be input by user of calculated by program) n - The number of states. (input by user)

next_state_file_ptr - A pointer to the beginning of the quantized data file. (corresponds to the based variable the_next_state_mapping)

next_state_map - A (num_state_combs x num_input_combs) array which contains the code of the next state for each state/input combination.

num_controllable_cells - The number of controllable cells

number_of_steps_i - A (p x 1) array containing the number of steps of the A/D converter for each input. (input by user)

number_of_steps_s - A (n x 1) array containing the number of steps of the A/D converter for each state. (input by user)

num_input_combs - The number of input combinations.

num_state_combs - The number of state combinations.

offset_i - A (p x 1) array used in computing the coded version of the input.

offset_s - A (n x 1) array used in computing the coded version of the state.

p - The modified number of inputs. This value is identical to the variable "p_real" above except in the case where "p_real" is zero in which the value of "p" becomes 1.

p_real - The number of inputs (input by user)

phi_matrix - A (n x n) array, the discrete - time system matrix (may be input by user or calculated by program)

quantum_step_size_i - A (p x 1) array containing the quantum steps size of the A/D converter for each input. It is used to convert the continuous - time arrays into discrete_time arrays and vice-versa.

quantum_step_size_s - A (n x 1) array containing the quantum step size of the A/D converter for each state. It is used to convert the continuous - time arrays into discrete_time arrays and vice-versa.

sat_edge - A (num_state_combs x num_input_combs) one bit array. The elements of the array are "1" if the corresponding cell is lead into saturation or to an uncontrollable cell given the corresponding input, and "0" otherwise.

state_cost_matrix - This (n x 1) array is the diagonal elements of the state weighting matrix (i.e. the "Q" matrix), which is assumed to be diagonal. It is entered by the user if a quadratic cost function for the controller is desired.

status flags - The following one bit variables which are used to record which part of the program has been completed for the current job - flag.cont_exists, flag.discrete_exists, flag.quantized_exists, flag.control_law_valid, and flag.sim_valid.

tau - The sampling period. This value is used to calculate the discrete - time model of the system. (input by user)

title - A user - inputted variable containing the title for the specific job. It may contain any keyboard characters and have a maximum length of 70 characters; however, if blanks are used, the entire variable must be enclosed in quotation marks (").

uncontrollable_cell - A one bit array of dimension (num_state_combs x 1). An uncontrollable cell is a cell which despite the given input will always lead to a saturated state. An element is "1" if the corresponding cell is an uncontrollable cell and a "0" if it is controllable.

voltage_lower_bound_i - A (n x 1) array containing the minimum voltage of the A/D converter for each input. (input by user)

voltage_lower_bound_s - A (n x 1) array containing the minimum voltage of the A/D converter for each state. (input by user)

voltage_upper_bound_i - A (n x 1) array containing the maximum voltage of the A/D converter for each input. (input by user)

voltage_upper_bound_s - A (n x 1) array containing the maximum voltage of the A/D converter for each state. (input by user)

3.0. FILES

Four files may be created during the execution of "DTQD."

3.1. job name.DATA

This file contains all of the above external variables which may be entered by the user, except job_name. The file is created via the subroutine CREATE_DATA_FILE of the procedure DTQD. Although the procedure CHANGE_PARAMETERS is designed to allow the user to enter or modify the data in this file, minor changes can be made easily using the text editor.

3.2. job name.NEXT STATE

This file is actually just a way of preserving the variable next_state_map. As stated in the previous section, this file contains the next_state for each state/input combination. The next state is stored in coded form as an integer and is retrieved via the coded state and input.

3.3. job name.CONTROL LAW

This file contains the optimal control law for the system. The file is in the form of a one-dimensional array of length equal to the number of cells (i.e. the number of state combinations). The control law is stored as an integer-coded input.

3.4. job name ts.PLOT

This file contains each state and input for every time interval that the system was simulated. It is this file that is used to make plots of the simulation.

4.0. PROCEDURES

The program is divided into six basic procedures, each part containing several sub-procedures. Figure B-1 is a flow diagram of the program which describes the interaction between these processes. Each of the six main routines as well as their respective internal subroutines are discussed in separate sections below.

4.1. DTQD

This procedure calls 10 subroutines, six of which are external procedures. It is one of the six basic sections of the entire program, the Main Menu. The purpose of this routine is to act as a menu so that the user can access the other five parts of the program. The internal subroutines, (CREATE_DATA_FILE, FREE_CONTR_EXTERN_VARS, SAVE_QUANT_FILE, and CLOSE_FILES), are called when the user is preparing to stop execution of the program.

4.1.1. CREATE_DATA_FILE. This internal subroutine is called by DTQD to save the data pertaining to the current job in a file named job_name.DATA. (See section 3.1 of this appendix) The variables are saved only if they have been allocated and set for the current job, either by accessing a previous data file or creating them in an appropriate routine. The variables which are always saved in this file are: title, flag.cont_exists, flag.discrets_exists, flag.quantized_exists, flag.control_law_valid, flag.sim_valid, flag.own_quant_file_exists. If any model of the system is valid or if a control law has been accessed, the variables n and p are saved. If a continuous model of the system exists, a_matrix and b_matrix, are recoreded in the file. Similarly, if a discrete_time model exists, phi_matrix, lambda_matrix, and tau are saved. If a quantized model exists, number_of_steps_s, number_of_steps_i, voltage_upper_bound_s, voltage_lower_bound_s, voltage_upper_bound_i, and voltage_lower_bound_i are saved. Finally, if a control law is valid for the current job, cost_function_code, state_cost_matrix, and input_cost_matrix are saved in the file job_name.data as well.

4.1.2. FREE_CONTR_EXTERN_VARS. This routine frees all of the controlled external variables used in the program.

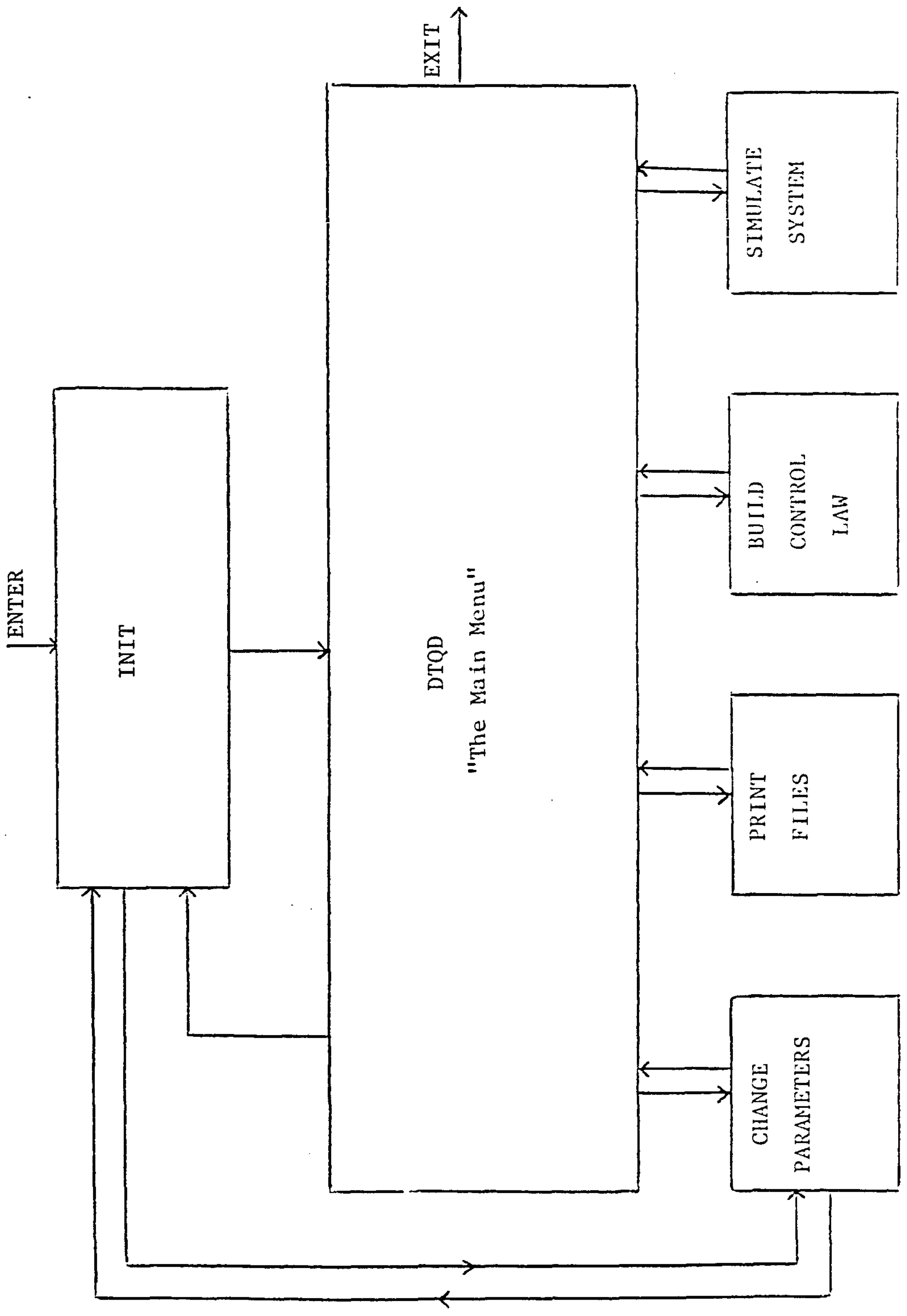


Figure B-1. Flow Diagram for "DTQD"

4.1.3. SAVE_QUANT_FILE. In this subroutine, the user has the opportunity to have the next - state array saved in a file. The advantage of having a file saved is that it need not be rebuilt, just read in, the next time that the job is accessed. However, if the file is very large, it may not be advantageous to have it take up so much space, and the user may opt to rebuild it each time. If the user does choose to have the array saved in a file, the variable flag.own_quant_file_exists is set to "1."

4.1.4. CLOSE_FILES. This suboutine closes the next - state and control law files by adjusting the bit count for the for the files job_name.next_state, and job_name.control_law.

4.2. INIT

The second basic part of the program is INIT. This procedure is called by DTQD when the program is initially executed and any time that the user opts to re-enter the initialization process. In this section the program prompts the user to enter the job name. The user can start a new job, access an old job, or modify an old job file. If the user accesses an old job file, GET_DATA_FILE is called. If the user starts a new job, GENERATE_PARAMETERS is called. If the user modifies an existing job, the data file from the old job is copied to create a new file and GET_DATA_FILE is called.

4.2.1. GENERATE_PARAMETERS. This subroutine prompts the user to enter the title for the job file, and then calls CHANGE_PARAMETERS.

4.2.2. GET_DATA_FILE. This subroutine is called to read in the data from the data file job_name.data. The title of the job or data file is printed on the screen and the user is asked if it is the correct file. If so, the data may be read in, depending on the value of the five status flags. Just as in CREATE_DATA_FILE (See section 4.1.1. of this appendix), if a certain model of the system has been created, or if a certain piece of the job has been completed, then the corresponding data may be read in. A subroutine of CHANGE_PARAMETERS called BUILD_MISC_ARRAYS is also called. Depending on the value of the variable flag.own_quant_file_exists, a file containing the next - state array is accessed or the subroutine of CHANGE_PARAMETERS, BUILD_NEXT_STATE_FILE, is called to generate the array. Also the procedure, BUILD_CONT_REG_SAT_EDG_ARRYS, another subroutine of CHANGE_PARAMETERS, is called.

4.3. CHANGE PARAMTERS

The third basic section of the program is the data modification section. In this procedure, the user can change the parameters of the continuous - time, discrete - time, and/or the quantized models of the system. This routine may be called by DTQD or by the subroutine of INIT, GENERATE_PARAMETERS.

Upon entering the program the user is asked which model is to be modified. If the continuous - time model is chosen, the user is asked which parameters of the model are to be changed. If the continuous - time model does not currently exist for the job, the program assumes that the user wants to create the continuous system and so the user will be prompted to enter all the parameters for the model.

If the discrete - time model is chosen to be modified, the user may change the parameters of the discrete system as can be done in the modification process of the continuous - time model. If, however, the continuous - time model for the system currently exists, the user can have the program generate the discrete - time model by asking the user to enter the sampling period, tau, and calling the subroutine BUILD_DISCRETE_MATRICIES.

If the user chooses to modify/create the quantized model of the system, the parameters of the A/D converter must be entered. Next, the subroutine BUILD_MISC_ARRAYS is called. The user is then given two choices: have the program generate the next - state array, or access a file containing a next - state array. The subroutine BUILD_CONT_REG_SAT_EDG_ARRYS is then called.

4.3.1. BUILD_DISCRETE_MATRICIES. This subroutine creates the discrete system matricies (phi_matrix and lambda_matrix) from the continuous - time matricies (a_matrix and b_matrix). The discrete - time system matrix, phi, is created by setting all the inputs and states equal to zero except the ith state which is set to 1. The value of the state after one time constant is then determined and the new state is set equal to the ith column of phi_matrix. To find the discrete - time input matrix, a similar procedure is followed. However, this time the ith input is set to 1 instead of the ith state. The change in state is found using the sixth order Runga-Kutta differential equation solver IMSL_DVERK.

4.3.2. BUILD_MISC_ARRAYS. This subroutine initializes the variables quantum_step_size_s, quantum_step_size_i, offset_s, offset_i, num_state_combs, and num_input_combs.

4.3.3. BUILD_CONT_REG_AND_SAT_EDG_ARRYS. This procedure builds the uncontrollable cell and saturated edge arrays. The variable num_controllable_cells is set to the number of controllable cells.

4.3.4. BUILD_NEXT_STATE_FILE. This procedure builds the quantized data array, next_state_map. The routine runs through every possible state and input combination, converts the state/input coded version to its discrete - time state and input arrays respectively, and determines the next state using the equation:

$$x(k+1) = \Phi\, x(k) + \Lambda\, u(k)$$

where,

$x(k)$ and $u(k)$ are the discrete - time state and input arrays respectively,

Φ is the discrete - time system matrix, phi_matrix,

Λ is the discrete - time input matrix, lambda_matrix.

Each next - state is checked for saturation. if saturation is found, the next state is converted to the coded form and is added to the array, next_state_map. Otherwise, a zero is added to the file signifying saturation.

4.4. PRINT IT

The fourth basic section is basically a menu which allows the user to examine various arrays and files. The subroutine DISPLAY_JOB_FILE, is called to display the parameters of the continuous- time model, discrete - time model, or the A/D converter. PRINT_NEXT_STATE_FILE and PRINT_CONTROL_LAW are called to display the next - state array and control law, respectively. CHECK_QUANTIZATION_LEVEL is called if the user wishes to have a check done on the quantization levels of the system.

4.4.1. DISPLAY_JOB_FILE. This procedure has not yet been written. However, when completed, it will allow the user to display any of the parameters saved in the data file job_name.data.

4.4.2. PRINT_NEXT_STATE_FILE. This procedure prints the next state code for each state/input combination.

4.4.3. CHECK_QUANTIZATION_LEVEL. This subroutine the user make a crude check on the quantization of the system. The check is done in two parts. The first is a summary of the cells moved from each state given a zero input. The number of cells moved in each direction and the total number of cells moved are computed and displayed. The second part of the report checks the number of cells moved from the zero state for each input at its smallest value. If the smallest value results in saturation, the smallest value which results in a non-saturated next state is used. The results are reported for each input, with the number of cells moved in each direction and the total number of cells moved being printed out. In this part, unlike the first, the cell movement is described by an absolute and average value. The absolute value is just the number of cells moved for each input. The average value is the absolute value divided by the number of steps between the smallest non-saturating input and the zero input.

After displaying the summary, the subroutine PRINT_SAT_EDGE_ARRAY is called and the user can have the saturation edge array printed. This array has the same matrix format as the next-state array, but the elements are displayed as either an "F" or a "T." A "T" is displayed if, given the corresponding input, the cell leads to saturation or to an uncontrollable cell. If not every cell is controllable, the user can print the uncontrollable cell array which will print the codes for each uncontrollable cell.

4.4.4. PRINT_CONTROL_LAW. This subroutine allows the user to print the coded form of the control law.

4.5. BUILD TOL REG AND CONT LAW

This is another basic section of the program. It is this procedure which builds the tolerant region and the control law.

4.5.1. BUILD_COST_FUNCTION. This is the first subroutine called if the user wishes to build a control law. The user is prompted to enter the desired cost function and if necessary the state and input weighting matricies. The user can use a minimum_time, minimum control effort, or quadratic cost function. If none of these are desired, an external file containing a control law may be accessed, or the user may write a routine containing a custom cost function for the control law to implement.

4.5.2. GET_TOLERANCES. In this procedure, the user is prompted to enter the tolerances necessary to find the tolerant region (center_cell_tolerance) and to compensate for edge irregularities (edge_cell tolerance).

4.5.3. INITIALIZE_CELL_STATUS_ARRAY. The array cell_status is initialized in this procedure. This array is one dimensional with length equal to num_state_combs. The procedure uses the variable edge_cell_tolerance set in GET_TOLERANCES to determine the "edge cells." The array is then initialized, giving each element one of the following values:

2:	if the cell is an edge cell
1:	if the cell is uncontrollable
0:	otherwise

As in section 5.2.4.13., the quantized model of a second order system may be thought of as a cell plane. Keeping this in mind, a typical second order system with a edge_cell_tolerance of one might have an initialized cell_status array ressembling the following:

1	2	2	2	1
2	0	0	0	2
2	0	0	0	2
2	0	0	0	2
1	2	2	2	1

4.5.4. INITIALIZE_CENTER_DIST_ARRAY. Another book-keeping array, center_dist, is initalized in this procedure. This routine uses the variable center_cell_tolerance which was set in GET_TOLERANCES to determine the tolerant region. The one dimensional array of length equal to the num_state_combs is then initalized. Each of the elements (i.e. state codes) is assigned a value equal to its distance from the origin. If this distance is greater than the center cell tolerance, however, the element is set equal to zero.

If the value of center_cell_tolerance was chosen to be two, the initialized center_dist array for a two dimensional system might look like:

0	0	0	0	0	0	0
0	2	2	2	2	2	0
0	2	1	1	1	2	0
0	2	1	0	1	2	0
0	2	1	1	1	2	0
0	2	2	2	2	2	0
0	0	0	0	0	0	0

4.5.5. FIND_ROOT_CELLS. This procedure finds the roots cells to create the tolerant region and control law. Each zero-valued element of the array cell_status is considered unmarked. This procedure marks each of the elements by implementing the following integer codes:

0:	cell is unmarked
1:	cell is uncontrollable
2:	cell is in the edge tolerant region
3:	cell is the zero state cell
4:	cell is reachable to a cell coded with 3
5:	cell is another root cell
6:	cell is reachable to a cell coded with 5
:	:
i:	cell is another root cell
i + 1:	cell is reachable to a cell coded with "i"

4.5.6. OPEN_CONTROL_LAW_FILE. This routine opens the control law file and initializes the control law array.

4.5.7. FIND_LOOPS_AND_CONTROL_LAW. In this procedure, the loops within each subtree (denoted by a separate root) are looked for within the center cell tolerant region. If a loop is found, the control law for the tolerant region is defined. If this can be done the procedure BUILD_OPTIMAL_CONTROL_LAW is called.

4.5.8. BUILD_OPTIMAL_CONTROL_LAW. This procedure builds the control law one cell at a time for the remainder of each of the subtrees by creating an optimal spanning tree based on the weighting matricies and cost function previously defined by the user.

4.6 SIMULATE SYSTEM

The procedure SIMULATE_SYSTEM simulates the closed loop system and implements the imbedding process. The user is first prompted to enter necessary parameters such as the number of imbedding levels, the scaling factor, the initial state and time, and the final time. Using the external subroutine OWN_SYS_TO_SIM.pl1, the user can simulate a continuous system which is different from the original system that the control law was developed for. The system used in OWN_SYS_TO_SIM must have at least as many states as the original system that the controller was designed for. If it has more states, the states which were initially used to develop the controller must be the first states of the new system.

The simulation starts with a check for saturation and controllability. Next, the magnitude of each state is studied and the proper imbedding level is evaluated by calling the subroutine FIND_REGION. If one sampling interval has elapsed, the control law is accessed to obtain the proper inputs. The control inputs are scaled to the proper size for the corresponding imbedding level. The sixth order Runga-Kutta differential equation solver IMSL_DVERK is then called to find the value of the state after one simulation step. The process is repeated until the final time is reached or until a state saturates. After the simulation is completed the subroutines BUILD_SIM_DATA_FILE and CHOOSE_YOUR_PLOT are then called.

4.6.1. BUILD_SIM_DATA_FILE. This subroutine puts the simulated data in a file title job_name_ts.plot if the user wishes. The simulation status flag, flag.sim_valid, is set only if the data is saved.

4.6.2. CHOOSE_YOUR_PLOT. Whether or not the simulation was successful, this subroutine is called and the user can plot the data. If the user wishes to make a graph, the program will ask for the other parameters to be entered. Any state and/or input, as well as time may be plotted on either axis. Also, more than one plot can be made using the same title, axis labels, and grid. Using the MULTICS procedures PLOT, PLOT_$SCALE, and PLOT_$SETUP, the program will proceed to plot the desired simulation data.

4.7. Miscellaneous Routines

Many of the procedures listed above call the following miscellaneous external subprograms: NUM_ANSWER_OK, YN_ANSWER_OK, and CONVERT_.

4.7.1. YN_ANSWER_OK. This procedure checks the response by the user whenever a yes/no answer is required. The routine will only accept "y", "yes", "n", or "no". If an incorrect response is entered, the program prompts the user to try again.

4.7.2. NUM_ANSWER_OK. This procedure checks the user's response whenever a menu selection is expected. The program only accepts an integer which represents a possible choice. If an incorrect response is entered, the user is asked to re-enter his choice.

4.7.3. CONVERT_. This procedure consists of six entries. An entry is called to convert the current representation of the state or input array into another representation. The arrays may be in a continuous, discrete, or coded form. The continuous version is that which has a range of lower_voltage_bound to upper_voltage_bound. The discrete form takes on distinct values in the range of 0 and number_of_steps for each state or input. Finally, the coded version gives each possible state combination and input combination a distinct integer code.

THIS PAGE LEFT BLANK INTENTIONALLY

APPENDIX C

PROGRAM LISTING FOR "DTQD"

THIS PAGE LEFT BLANK INTENTIONALLY

1.0 PROGRAM LISTING FOR "DTQD"

The following pages contain the pl/1 code for the program DTQD. The listings are organized into six basic procedures as discussed in Appendix B. DTQD is first followed by init, change_parameters, print_it, build_tol_reg_and_cont_law, simulate_system, and finally some miscellaneous routines.

For reference, the above procedures as well as their major subroutines are listed below in alphabetical order with corresponding page numbers.

```
DTQD: procedure options (main);

   dcl choice fixed;
   dcl choice_char character (1);
   dcl range fixed;
   dcl done bit(1);
   dcl first_init_flag bit(1);

   dcl sysin file;
   dcl sysprint file;
   dcl data_file file;

   dcl num_answer_ok entry (character (1), fixed, fixed);
   dcl init entry (bit(1), file);
   dcl print_it entry;
   dcl change_parameters entry;
   dcl build_tol_reg_and_cont_law entry;
   dcl simulate_system entry;

first_init_flag = "1"b;
call init (first_init_flag, data_file);
done = "0"b;
do while (done = "0"b);
   put edit ("1) Initialize") (skip, a);
   put edit ("2) Modify Data File") (skip, a);
   put edit ("3) Print Files") (skip, a);
   put edit ("4) Build Control Law") (skip, a);
   put edit ("5) Simulate") (skip, a);
   put edit ("6) Quit") (skip, a);
   put edit ("Enter choice  ==>  ") (skip (2), a);
   get list (choice_char);
   range = 6;
   call num_answer_ok (choice_char, range, choice);
   goto case (choice);
      case(1):  if (first_init_flag = "0"b) then do;
                   call save_quant_file;
                   call create_data_file;
                end;
               call init (first_init_flag, data_file);
      goto end_case;
      case (2): call change_parameters;
      goto end_case;
      case (3): call print_it;
      goto end_case;
      case (4): call build_tol_reg_and_cont_law;
      goto end_case;
      case (5): call simulate_system;
      goto end_case;
      case (6): done = "1"b;
   end_case:
end; /* while */
if (first_init_flag = "0"b) then do;
    call save_quant_file;
   call create_data_file;
end;
call free_contr_extern_vars;
call close_files;

create_data_file: procedure;

   dcl job_name character (50) varying external;
   dcl title character (70) varying external;

   dcl true bit(1)  initial ("1"b);
   dcl false bit(1)  initial ("0"b);
   dcl 1 flag external,
          2 cont_exists bit(1),
          2 discrete_exists bit(1),
          2 quantized_exists bit(1),
          2 control_law_valid bit(1),
          2 sim_valid bit(1),
          2 own_quant_file_exists bit(1);

   dcl next_state_file_ptr pointer external;
   dcl own_quant_data_title character (70)  external;

   dcl n fixed external;
   dcl p_real fixed external;
   dcl p fixed external;

   dcl a_matrix (1:n, 1:n) float controlled external;
   dcl b_matrix (1:n, 1:p) float controlled external;

   dcl tau float external;
   dcl phi_matrix (1:n, 1:n) float controlled external;
   dcl lambda_matrix (1:n, 1:p) float controlled external;

   dcl number_of_steps_s (1:n) fixed controlled external;
   dcl voltage_upper_bound_s (1:n) float controlled external;
   dcl voltage_lower_bound_s (1:n) float controlled external;

   dcl number_of_steps_i (1:p) fixed controlled external;
   dcl voltage_upper_bound_i (1:p) float controlled external;
   dcl voltage_lower_bound_i (1:p) float controlled external;

   dcl cost_function_code fixed external;
   dcl state_cost_matrix (1:n)  float controlled external;
   dcl input_cost_matrix (1:p) float controlled external;

   dcl data_file file;

   /* The above variables are stored in the same order as decla
red */

   dcl skip_amount fixed;
   dcl i fixed;
   dcl j fixed;

   /* **** */


   open file (data_file) title ("vfile_ "||job_name||".data") s
tream output;


   /*  *** */


   put file (data_file) edit (title)(skip, a(70));

   put file (data_file) edit (flag.cont_exists)(skip, b(1));
   put file (data_file) edit (flag.discrete_exists)(skip, b(1))
;
   put file (data_file) edit (flag.quantized_exists)(skip, b(1)
);
   put file (data_file) edit (flag.control_law_valid)(skip, b(1
));
   put file (data_file) edit (flag.sim_valid)(skip, b(1));
   put file (data_file) edit (flag.own_quant_file_exists)(skip,
 b(1));

   if (flag.cont_exists = true|flag.discrete_exists = true|flag
.quantized_exists = true|flag.control_law_valid = true|flag.si
m_valid = true) then
      do;
          put file (data_file) edit (n)(skip,f(5));
```

```
            put file (data_file) edit (p)(skip, f(5));
        end;

    if (flag.cont_exists = true) then
        do;
            do i = 1 to n;
                do j = 1 to n;
                    put file (data_file) edit (a_matrix (i,j))(skip,
 f(12, 4));
                end;
            end;
            do i = 1 to n;
                do j = 1 to p;
                    put file (data_file) edit (b_matrix (i,j))(skip,
 f(12, 4));
                end;
            end;
        end;

    if (flag.discrete_exists = true) then
        do;
            put file (data_file) edit (tau)(skip, f(12, 4));
            do i = 1 to n;
                do j = 1 to n;
                    put file (data_file) edit (phi_matrix (i, j))(sk
ip, f(12, 4));
                end;
            end;
            do i = 1 to n;
                do j = 1 to p;
                    put file (data_file) edit (lambda_matrix (i, j))
(skip, f(12, 4));
                end;
            end;
        end;

    if (flag.quantized_exists = true | flag.own_quant_file_exist
s = true) then
        do;
            do i = 1 to n;
                put file (data_file) edit (number_of_steps_s (i))(s
kip, f(5));
            end;
            do i = 1 to n;
                put file (data_file) edit (voltage_upper_bound_s(i)
, voltage_lower_bound_s(i))
                                        (skip, f(12, 4), x(3), f(12, 4));
            end;
            do i = 1 to p;
                put file (data_file) edit (number_of_steps_i (i))(s
kip, f(5));
            end;
            do i = 1 to p;
                put file (data_file) edit (voltage_upper_bound_i (i
), voltage_lower_bound_i (i))
                                                  (skip, f(12, 4), x(3), f(1
2, 4));
            end;

    if (flag.own_quant_file_exists = true) then
        put file (data_file) edit (own_quant_data_title)(skip, a(
70));
    else
        put file (data_file) skip;

        end;
```

```
          if (flag.control_law_valid = true) then
             do;
                put file (data_file) edit (cost_function_code) (
skip, f(5));
                if (cost_function_code = 2) then
                   do;
                      dc i = 1 to n;
                         put file (data_file) edit (state_cost_m
atrix (i))
                                                      (skip, f(12,4)
);
                      end;
                   end;
                if (cost_function_code = 2 | cost_function_code
= 3) then
                   do;
                      do i = 1 to p;
                         put file (data_file) edit (input_cost_m
atrix(i))
                                                      (skip, f(12,4)
);
                      end;
                   end;
             end;

   close file (data_file);

    end create_data_file;
free_contr_extern_vars: procedure;

   dcl n fixed external;
   dcl p fixed external;
   dcl num_state_combs fixed external;
   dcl num_input_combs fixed external;
   dcl number_of_steps_s (1:n) fixed controlled external;
   dcl number_of_steps_i (1:p) fixed controlled external;
   dcl offset_s (1:n) fixed controlled external;
   dcl offset_i (1:p) fixed controlled external;
   dcl quantum_step_size_s (1:n) fixed controlled external;
   dcl quantum_step_size_i (1:p) fixed controlled external;

   dcl voltage_upper_bound_s (1:n) float controlled external;
   dcl voltage_lower_bound_s (1:n) float controlled external;
   dcl voltage_upper_bound_i (1:p) float controlled external;
   dcl voltage_lower_bound_i (1:p) float controlled external;
   dcl phi_matrix (1:n, 1:n) float controlled external;
   dcl lambda_matrix (1:n, 1:p) float controlled external;

   dcl next_state_map (1:num_state_combs, 1:num_input_combs) fi
xed controlled external;

   dcl uncontrollable_cell (1:num_state_combs) bit(1) controlle
d external;
   dcl sat_edge (1:num_state_combs, 1:num_input_combs) bit(1) c
ontrolled external;

   free number_of_steps_s, number_of_steps_i, offset_s, offset_
i;
   free quantum_step_size_s, quantum_step_size_i;

   free voltage_upper_bound_s, voltage_lower_bound_s;
   free voltage_upper_bound_i, voltage_lower_bound_i;
   free phi_matrix, lambda_matrix;
   free next_state_map;
   free uncontrollable_cell, sat_edge;

end free_contr_extern_vars;
```

```
save_quant_file: procedure;

   dcl next_state_file_ptr pointer external;
   dcl num_state_combs fixed external;
   dcl num_input_combs fixed external;
   dcl next_state_map (1:num_state_combs, 1:num_input_combs) fi
xed controlled external;
   dcl the_next_state_mapping (1:num_state_combs, 1:num_input_c
ombs) fixed binary(18)  unsigned based (next_state_file_ptr);
   dcl job_name character(50) varying external;
  dcl own_quant_data_title character (70)  external;

   dcl true bit(1)  initial ("1"b);
   dcl false bit(1)  initial ("0"b);
   dcl 1 flag external,
           2 cont_exists bit(1),
           2 discrete_exists bit(1),
           2 quantized_exists bit(1),
           2 control_law_valid bit(1),
           2 sim_valid bit(1),
           2 own_quant_file_exists bit(1);

   dcl working_dir character(168) external;
   dcl bit_count fixed bin(24);
   dcl code fixed bin(35);
   dcl answer character(3) varying;
   dcl i fixed;
   dcl j fixed;

   dcl hcs_$initiate_count entry (char(*),char(*),char(*), fixe
d bin(24),
                                   fixed bin(2), ptr, fixed bin(
35));
   dcl hcs_$make_seg entry (char(*), char(*), char(*), fixed bi
n(5),
                                ptr, fixed bin(35));
   dcl delete entry options (variable);

   dcl yn_answer_ok entry (character(3) varying);
   dcl sysin file input;
   dcl sysprint file output;


   if (flag.quantized_exists = true) then do;
   put edit("Would you like to save the quantized state file? =
> ")(skip,a);
   get list (answer);
   call yn_answer_ok (answer);
   if (answer = "y" | answer = "yes") then
      do;
        if (flag.own_quant_file_exists = false) then
          own_quant_data_title = job_name||".next_state";

         call hcs_$initiate_count (working_dir, own_quant_data_
title, "",
                                  bit_count, 0, next_state_fil
e_ptr, code);
         call delete (own_quant_data_title, "-bf");
         call hcs_$make_seg (working_dir, own_quant_data_title,
 "",
                              01010b, next_state_file_ptr, code)
;

         do i = 1 to num_state_combs;
            do j = 1 to num_input_combs;
               the_next_state_mapping(i,j) = next_state_map (i,
j);
```

```
              end;
           end;
          flag.own_quant_file_exists = true;
         end;
     end;

   end save_quant_file;
   close_files: procedure;

     dcl job_name character (50) varying external;

     dcl 1 flag external,
              2 cont_exists bit(1),
              2 discrete_exists bit(1),
              2 quantized_exists bit(1),
              2 control_law_valid bit(1),
              2 sim_valid bit(1),
              2 own_quant_file_exists bit(1);

     dcl adjust_bit_count entry options (variable);

     if (flag.quantized_exists= "1"b) then
     call adjust_bit_count (job_name||".next_state", "-ch");

     if (flag.control_law_valid = "1"b) then
     call adjust_bit_count (job_name||".control_law", "-ch");

   end close_files;


   end DTQD;
```

```
init: procedure (first_init_flag, data_file);

  dcl first_init_flag bit(1);
  dcl data_file file;


  dcl job_name character (50) varying external;
  dcl working_dir character (168) external;

  dcl next_state_file_ptr pointer external;
  dcl control_law_file_ptr pointer external;

  dcl 1 flag static external,
        2 cont_exists bit(1),
        2 discrete_exists bit(1),
        2 quantized_exists bit(1),
        2 control_law_valid bit(1),
        2 sim_valid bit(1),
        2 own_quant_file_exists bit(1);


  dcl true bit(1) initial ("1"b);
  dcl false bit(1) initial ("0"b);
  dcl 1 flag2,
        2 done_init bit(1),
        2 build_mode bit(1),
        2 good_job_name bit(1);

  dcl good_job_title bit(1);

  dcl choice fixed;
  dcl c character (1);
  dcl range fixed;
  dcl job_name_new character (50) varying;
  dcl answer character (3) varying;
  dcl bit_count fixed bin(24);
  dcl code fixed bin(35);

  dcl sysin file input;
  dcl sysprint file output;

  dcl null builtin;

  dcl undefinedfile condition;

  dcl num_answer_ok entry (character (1), fixed, fixed);
  dcl print_it entry;
  dcl change_parameters entry;
  dcl yn_answer_ok entry (character (3) varying);

  dcl hcs_$initiate_count entry (char(*), char(*), char(*),
                  fixed bin(24), fixed bin(2), ptr, fixed bin
(35));
  dcl copy entry options (variable);
  dcl get_wdir_ entry returns (character (168));
  dcl convert_status_code_ entry (fixed bin (35), char (8) ali
gned,
                                      char (100) aligned);

  on undefinedfile (data_file) flag2.good_job_name = false;

  working_dir = get_wdir_ ();
  flag2.done_init = false;
  do while ( done_init = false );
    good_job_title = true;
    flag2.good_job_name = true;
    put edit ("Would you like to :")(skip, a);
    put skip;
```

```
     put edit ("1.  Access an old job file")(skip,a);
     put edit ("2.  Create a new job file ")(skip,a);
     put edit ("3.  Modify an old job file")(skip,a);
     put edit ("4.  Return to Main Menu")(skip,a);
     put skip;
     put edit ("Please choose one of the above => ")(skip,a);
     get list (c);
     range = 4;
     call num_answer_ck (c, range, choice);

      if (choice = 1) then
         do;
            flag2.builc_mode = false;
            put edit ("Enter the job name => ")(skip, a);
            get list (job_name);
            open file (data_file) title ("vfile_ "||job_name||"
.data")
                                                          stream i
nput;
            if (flag2.good_job_name = true) then do;
               call get_data_file (data_file, good_job_title);
               put edit("The current status of this job is: ")(
skip,a);
               put skip;
               if (flag.cont_exists = true) then
                  put edit ("    A continuous system exists")(s
kip,a);
               if (flag.discrete_exists = true) then
                  put edit ("    A discrete system exists")(ski
p,a);
               if (flag.quantized_exists = true) then
                  put edit ("    A quantized system exists")(sk
ip,a);
               if (flag.control_law_valid = true) then
                  put edit ("    A control law is valid ")(skip
,a);
               if (flag.sim_valid = true) then
                  put edit ("    A simulation of the job exists
")(skip,a);
               if (flag.cont_exists = false & flag.discrete_exi
sts = false &
                         flag.quantized_exists = false & flag.contr
ol_law_valid = false &
                             flag.sim_valid = false) then
                  put edit ("   No models or files exist for th
is job")(skip,a);
               put skip;
            end;
         end;
      if (choice = 2) then
         do;
            put edit ("Enter name of the new job file => ")(ski
p,a);
            get list (job_name);
            flag2.build_mode = true;
            call generate_parameters;
         end;
      if (choice = 3) then
         do;
            put edit ("Enter name of job file to be modified =>
 ")(skip, a);
            get list (job_name);
            put edit ("Enter name of new job file => ")(skip, a
);
            get list (job_name_new);
            open file (data_file) title ("vfile_ "||job_name||"
.data") stream input;
            flag2.build_mode = false;
```

```
                if (flag2.good_job_name = true) then
                   do;
   /***/              call copy (job_name||".data", job_name_new||"
.data", "-bf");
                      open file (data_file) title ("vfile_ "||job_n
ame_new
                                                  ||".data") stream
input;
                      call get_data_file (data_file, good_job_title
);
                      if (control_law_valid = true) then do;
                        call hcs_$initiate_count (working_dir, job_
name||".control_law",
                            "",bit_count, 0, control_law_file_ptr,
code);
                      end;
                      job_name = job_name_new;
                    end;
             end;
         if ( choice = 4) then
              flag2.done_init = true;
         if ( flag2.good_job_name = false ) then
             do;
                 put edit(job_name, ".data does not exist.")(skip,
 a, a);
                 flag2.done_init = false;
             end;
         if ( good_job_title = false ) then
            do;
               flag2.good_job_name = false;
               flag2.done_init = false;
            end;
         if (flag2.good_job_name = true & choice < 4 ) then
            do;
               first_init_flag = false;
               flag2.done_init = true;
             end;
   end;

generate_parameters:  procedure;

   dcl 1 flag external,
           2 cont_exists bit(1),
           2 discrete_exists bit(1),
           2 quantized_exists bit(1),
           2 control_law_valid bit(1),
           2 sim_valic bit(1),
           2 own_quant_file_exists bit(1);

   dcl title character (70) varying external;

   dcl true bit(1) initial ("1"b);
   dcl false bit(1) iritial ("0"b);
   dcl sysin file input;
   dcl sysprint file output;
   dcl data_file file;

   dcl change_parameters entry;


   flag.cont_exists = false;
   flag.discrete_exists = false;
   flag.quantized_exists = false;
   flag.control_law_valid = false;
   flag.sim_valid = false;
   flag.own_quant_file_exists = false;

   put skip;
```

```
    put edit("Enter a title for the data file ")(skip,a);
    put edit ("Note:  Quotes are required if more than one word
is used")(skip, a);
    put skip;
    get list (title);

    call change_parameters;

end generate_parameters;
get_data_file: procedure (data_file, good_job_title);

    dcl data_file file;
    dcl good_job_title bit(1);

    dcl job_name character (50) varying external;
    dcl title character (70) varying external;

    dcl true bit(1)  initial ("1"b);
    dcl false bit(1)  initial ("0"b);
    dcl 1 flag static external,
            2 cont_exists bit(1),
            2 discrete_exists bit(1),
            2 quantized_exists bit(1),
            2 control_law_valid bit(1),
            2 sim_valid bit(1),
            2 own_quant_file_exists bit(1);


    dcl next_state_file_ptr pointer external;
    dcl own_quant_data_title character (50)  external;

    dcl n fixed external;
    dcl p_real fixed external;
    dcl p fixed external;

    dcl a_matrix (1:n, 1:n) float controlled external;
    dcl b_matrix (1:n, 1:p) float controlled external;

    dcl tau float external;
    dcl phi_matrix (1:n, 1:n) float controlled external;
    dcl lambda_matrix (1:n, 1:p) float controlled external;

    dcl number_of_steps_s (1:n) fixed controlled external;
    dcl voltage_upper_bound_s (1:n) float controlled external;
    dcl voltage_lower_bound_s (1:n) float controlled external;

    dcl number_of_steps_i (1:p) fixed controlled external;
    dcl voltage_upper_bound_i (1:p) float controlled external;
    dcl voltage_lower_bound_i (1:p) float controlled external;

    dcl num_state_combs fixed external;
    dcl num_input_combs fixed external;

    dcl the_next_state_mapping (1:num_state_combs, 1:num_input_c
ombs) fixed binary (18)
                                        unsigned based (next_state_file
_ptr);
    dcl next_state_map (1:num_state_combs, 1:num_input_combs) fi
xed controlled external;

    dcl control_law_file_ptr pointer external;
    dcl cost_function_code fixed external;
    dcl state_cost_matrix (1:n) float controlled external;
    dcl input_cost_matrix (1:p) float controlled external;


    /* The above variables are stored in the same order as decla
red */
```

```
  dcl still_data_left bit(1);
  dcl skip_amount fixed;
  dcl i fixed;
  dcl j fixed;
  dcl answer character (3) varying;

  dcl sysin file input;
  dcl change_parameters$build_next_state_file entry;
  dcl change_parameters$build_misc_arrays entry;
  dcl change_parameters$build_cont_reg_sat_edg_arrys entry;
  dcl change_parameters entry;
  dcl yn_answer_ok entry (character(3) varying);
  dcl sysprint file output;

  dcl undefinedfile condition;
  dcl working_dir character (168) external;
  dcl bit_count fixed bin(24);
  dcl code fixed bin(35);
  dcl hcs_$initiate_count entry (char(*), char(*), char(*), fi
xed bin(24),
                          fixed bin(2), ptr, fixed bin (35));

  /* **** */


  get file (data_file) edit (title)(skip, a(70));
  put edit ("The title of this data file is: ")(skip,a);
  put edit (title)(skip,x(3), a);
  put edit ("Is this the correct file?  ")(skip,a);
  get list (answer);
  call yn_answer_ok (answer);
if (answer = "y" | answer = "yes") then
   do;
        get file (data_file) edit (flag.cont_exists)(skip, b(1
));
        get file (data_file) edit (flag.discrete_exists)(skip,
 b(1));
        get file (data_file) edit (flag.quantized_exists)(skip
, b(1));
        get file (data_file) edit (flag.control_law_valid)(ski
p, b(1));
        get file (data_file) edit (flag.sim_valid)(skip, b(1))
;
        get file (data_file) edit (flag.own_quant_file_exists)
(skip, b(1));


  if (flag.cont_exists = true|flag.discrete_exists = true|flag
.quantized_exists = true |flag.control_law_valid = true|flag.s
im_valid = true) then
  do;
  get file (data_file) edit (n)(skip,f(5));
  get file (data_file) edit (p_real)(skip, f(5));
  if (p_real = 0) then
     p = 1;
  else
     p = p_real;
  end;


  if (flag.cont_exists = true) then
      do;
        allocate a_matrix;
        allocate b_matrix;
        do i = 1 to n;
           do j = 1 to n;
```

```
                get file (data_file) edit (a_matrix (i,j))(skip,
 f(12, 4));
                end;
            end;
            do i = 1 to n;
               do j = 1 to p;
                get file (data_file) edit (b_matrix (i,j))(skip,
 f(12, 4));
               end;
            end;
         end;

   if (flag.discrete_exists = true) then
         do;
            allocate phi_matrix, lambda_matrix;
            get file (data_file) edit (tau)(skip, f(12, 4));
            do i = 1 to n;
               do j = 1 to n;
                get file (data_file) edit (phi_matrix (i, j))(sk
ip, f(12, 4));
               end;
            end;
            do i = 1 to n;
               do j = 1 to p;
                get file (data_file) edit (lambda_matrix (i, j))
(skip, f(12, 4));
               end;
            end;
         end;

   if (flag.quantized_exists = true | flag.own_quant_file_exist
s = true) then
         do;
            allocate number_of_steps_s, voltage_upper_bound_s, vol
tage_lower_bound_s;
            allocate number_of_steps_i, voltage_upper_bound_i, vol
tage_lower_bound_i;

            do i = 1 to n;
               get file (data_file) edit (number_of_steps_s (i))(s
kip, f(5));
            end;
            do i = 1 to n;
               get file (data_file) edit (voltage_upper_bound_s(i)
, voltage_lower_bound_s(i))
                                   (skip, f(12, 4), x(3), f(12, 4));
            end;
            do i = 1 to p;
               get file (data_file) edit (number_of_steps_i (i))(s
kip, f(5));
            end;
            do i = 1 to p;
               get file (data_file) edit (voltage_upper_bound_i (i
), voltage_lower_bound_i (i))
                                             (skip, f(12, 4), x(3), f(1
2, 4));
            end;

            call change_parameters$build_misc_arrays;


   if (flag.own_quant_file_exists = true) then do;
        get file (data_file) edit (own_quant_data_title)(skip, a(
70));
        call hcs_$initiate_count (working_dir, own_quant_data_tit
le,
                              "", bit_count, 0, next_state_file_ptr, c
ode);
```

```
        allocate next_state_map;
        do i = 1 to num_state_combs;
           do j= 1 to num_input_combs;
              next_state_map (i,j) = the_next_state_mapping(i,j);
           end;
        end;
     end;
     else do;
       call change_parameters$build_next_state_file;
     end;

           call change_parameters$build_cont_reg_sat_edg_arrys;

        end;

           if (flag.control_law_valid = true) then
             do;
                call hcs_$initiate_count (working_dir, job_name||
".control_law", "",
                          bit_count, 0, control_law_file_ptr, co
de);

                get file (data_file) edit (cost_function_code) (s
kip, f(5));
                if (cost_function_code = 2) then
                   do;
                         allocate state_cost_matrix;
                         do i = 1 to n;
                         get file (data_file) edit (state_cost_ma
trix (i))(skip, f(12,4));
                      end;
                   enc;
                 if (cost_function_code = 2 | cost_function_code
= 3 ) then
                   do;

                        allocate input_cost_matrix;
                         do i = 1 to p;
                           get file (data_file) edit (input_cost_
matrix (i))(skip, f(12,4));
                        end;
                     end;

            end;
    end;
    else
        good_job_title = false;

     close file (data_file);

      end get_data_file;

end init;
```

```
change_parameters: procedure;

   dcl job_name character (50) varying external;
   dcl working_dir character (168) external;
   dcl next_state_file_ptr pointer external;

   dcl n fixed external;
   dcl p fixed external;
   dcl p_real fixed external;
   dcl number_of_steps_s (1:n) fixed controlled external;
   dcl number_of_steps_i (1:p) fixed controlled external;

   dcl voltage_upper_bound_s (1:n) float controlled external;
   dcl voltage_lower_bound_s (1:n) float controlled external;
   dcl voltage_upper_bound_i (1:p) float controlled external;
   dcl voltage_lower_bound_i (1:p) float controlled external;
   dcl a_matrix (1:n, 1:n) float controlled external;
   dcl b_matrix (1:n, 1:p) float controlled external;
   dcl tau float external;
   dcl phi_matrix (1:n, 1:n) float controlled external;
   dcl lambda_matrix (1:n, 1:p) float controlled external;
   dcl offset_s (1:n) fixed controlled external;
   dcl offset_i (1:p) fixed controlled external;
   dcl num_state_combs fixed external;
   dcl num_input_combs fixed external;
   dcl quantum_step_size_s (1:n) float controlled external;
   dcl quantum_step_size_i (1:p) float controlled external;
   dcl num_controllable_cells fixed external;
   dcl uncontrollable_cell (1:num_state_combs) bit(1) controlle
d external;
   dcl sat_edge (1:num_state_combs, 1:num_input_combs) bit(1) c
ontrolled
                 external;

   dcl the_next_state_mapping (1:num_state_combs, 1:num_input_c
ombs) fixed binary (18)
                      unsigned based (next_state_file_ptr);
   dcl next_state_map (1:num_state_combs, 1:num_input_combs) fi
xed controlled external;

   dcl title character (70) varying external;
   dcl own_quant_data_title character (70) external;

   dcl bit_count fixed bin(24);
   dcl code fixed bin(35);

   dcl i fixed;
   dcl j fixed;
   dcl answer character (3) varying;
   dcl c character (1);
   dcl choice fixed;
   dcl c2 character(1);
   dcl choice2 fixed;
   dcl c3 character (1);
   dcl choice3 fixed;
   dcl range fixed;
   dcl col_min fixed;
   dcl col_max fixed;


   dcl true bit(1)  initial ("1"b);
   dcl false bit(1)  initial ("0"b);
   dcl 1 flag external,
         2 cont_exists bit(1),
         2 discrete_exists bit(1),
         2 quantized_exists bit(1),
         2 control_law_valid bit(1),
         2 sim_valid bit(1),
            2 own_quant_file_exists bit(1);

   dcl 1 flag2,
            2 changed_n bit(1),
            2 need_set bit(1),
            2 modify_discrete bit(1),
            2 file_not_exist bit (1);


   dcl data_file file;
   dcl sysin file input;
   dcl sysprint file output;

   dcl convert_$cont_state_to_dis_state entry ((*) float, (*) f
ixed);
   dcl convert_$dis_state_to_code entry entry ((*) fixed, fixed
);
   dcl convert_$code_to_dis_state entry (fixed, (*)fixed);
   dcl convert_$dis_state_to_cont_state entry ((*) fixed, (*) f
loat);
   dcl convert_$code_to_dis_input entry (fixed, (*) fixed);
   dcl convert_$dis_input_to_cont_input entry ((*) fixed, (*) f
loat);
   dcl num_answer_ok entry (character (1), fixed, fixed);
   dcl yn_answer_ok entry (character (3) varying);
   dcl print_next_state_file entry;

   dcl hcs_$initiate_ccunt entry (char(*), char(*), char(*), fi
xed bin (24),
                                             fixed bin (2), ptr, fixed bi
n (35));
   dcl hcs_$make_seg entry (char(*), char(*), char(*), fixed bi
n (5),
                                              ptr, fixed bin(35));
   dcl delete entry options (variable);
   dcl copy entry opticns (variable);

   dcl null builtin;


PARMS:  put skip;
        flag2.modify_ciscrete = false;
        put edit ("Which of the following would you like to mo
dify/create?")
                                                             (skip,
a);
        put skip;
        put edit ("1.  Title of the job file")(skip, a);
        put edit ("2.  Continuous system parameters")(skip,a);
        put edit ("3.  Discrete system parameters")(skip, a);
        put edit ("4.  Quantized system parameters")(skip, a);
        put edit ("5.  None of the above")(skip, a);
        put skip;
        put edit (" Please choose one => ")(skip,a);

        get list (c);
        range = 5;
        call num_answer_ok (c, range, choice);
        flag2.need_set = false;

        goto case(choice);

case(1):   put skip;
           put edit ("Enter a file title ")(skip, a);
           put edit ("Note:  Quotes are required if more than o
ne word.")(skip,a);
           put skip;
```

```
          get list (title);
          goto PARMS;

case(2):  put skip;
          if (flag.ccrt_exists = true) then
             do;
                put ecit ("Which parameter(s) would you like
                                              to change?")(s
kip, a);
                put edit ("1.  Number of states")(skip,a);
                put edit ("2.  Number of inputs")(skip,a);
                put edit ("3.  System matrix, A")(skip,a);
                put ecit ("4.  Input matrix, B")(skip,a);
                put edit ("5.  All of the above")(skip,a);
                put edit ("6.  None of the above")(skip,a);
                put skip;
                put edit ("Please choose one => ")(skip,a);
                get list (c2);
                range = 6;
                call num_answer_ok (c2, range, choice2);

                if (choice2 ^= 6) then
                   flag2.need_set = true;
                goto case_2(choice2);
             end;
          else
             do;
                flag.cont_exists = true;
                flag2.need_set = true;
                choice2 = 5;
                allocate a_matrix, b_matrix;
                goto case_2(1);
             end;

     case_2(1):  put skip;
                 put edit ("Enter number of states => ")(skip,
 a);
                 get list (n);
                 if (choice2 ^= 5) then
                    do;
                    if (flag2.modify_discrete = true) then
                       put edit ("The discrete system and inpu
t matricies must now be modified:")(skip,a);
                    else
                       put edit("The system matrix, A, and inp
ut matrix, B, must now be modified:")(skip,a);
                    flag2.changed_n = true;
                    goto case_2(3);
                    end;
                 else
                    goto case_2(2);

     case_2(2):  put skip;
                 put edit ("Enter number of inputs => ")(skip,
 a);
                 get list (p_real);
                 if (p_real > 0) then
                    p = p_real;
                 else
                    p = 1;
                 if (choice2 ^= 5 ) then
                    do;
                       if (flag2.modify_discrete = true) then
                          put edit ("The discrete input matri
x must now be modified:")(skip, a);
                       else
                          put edit("The input matrix, B, must
 now be modified:")(skip, a);
```

```
                                goto case_2(4);
                             end;
                          else
                             goto case_2(3);

          case_2(3):      put skip;
                          if (flag2.modify_discrete = true) then do;
                             free phi_matrix;
                             allocate phi_matrix;
                             put edit ("Enter values for the discrete s
ystem matrix, phi")(skip,a);
                          end;
                          else do;
                             free a_matrix;
                             put edit ("Enter values for the A matrix")
(skip, a);
                             allocate a_matrix;
                          end;
                          put skip;
                          do i = 1 to n;
                             do j = 1 to n;
                                if (flag2.modify_discrete = true) then
do;
                                   put edit ("phi (", i, ", ", j, ") =>
 ")(x(3), a, f(3), a, f(3), a);
                                   get list (phi_matrix(i, j));
                                end;
                                else do;
                                   put edit ("A (", i, ", ", j, ") => "
)(x(3), a, f(3), a, f(3), a);
                                  get list (a_matrix(i, j));
                               end;
                             end;
                          end;
                          if (choice2 ^= 5 ) then do;
                             if (flag2.changed_n = true) then
                                do;
                                   flag2.changed_n = false;
                                   goto case_2(4);
                                end;
                             else do;
                                if (flag2.modify_discrete = true) then
                                   goto Modify_dis;
                                else
                                   goto case(2);
                             end;
                           end;
                          else
                             goto case_2(4);

          case_2(4):      put skip;
                          if ( p_real > 0 ) then
                             do;
                                if (flag2.modify_discrete = true) then
                                   do;
                                      free lambda_matrix;
                                      allocate lambda_matrix;
                                      put edit ("Enter values for the d
iscrete input matrix, lambda")(skip,a);
                                   end;
                                else
                                   do;
                                      free b_matrix;
                                      allocate b_matrix;
                                      put edit ("Enter values for the B
 matrix")(skip, a);
                                   end;
                                put skip;
```

```
                                        do i = 1 to n;
                                            do j = 1 to p;
                                                if (flag2.modify_discrete = true)
 then do;
                                                    put edit ("lambda (", i, ", ",
 j, ") => ")
                                                     (x(4), a, f(3), a, f(3), a);
                                                    get list (lambda_matrix(i, j))
;
                                                end;
                                                else do;
                                                    put edit ("B (", i, ", ", j, "
) => ")(x(4), a, f(3), a, f(3), a);
                                                    get list (b_matrix(i, j));
                                                end;
                                            end;
                                        end;
                                    end;
                                else
                                    do;
                                        free b_matrix;
                                        allocate b_matrix;
                                        b_matrix = 0;
                                    end;
                                if (flag2.modify_discrete = true) then
                                    goto Modify_dis;
                                else
                                    goto case(2);

        case_2(5):              goto case_2(1);

        case_2(6):               if (flag2.need_set = true) then
                                  do;
                                      flag.cont_exists = true;
                                      flag.discrete_exists = false;
                                      flag.quantized_exists = false;
                                      flag.control_law_valid = false;
                                      flag.sim_valid = false;
                                  end;
                                 goto PARMS;


case(3):   /* Discrete Parms */
           put skip;
           flag2.modify_discrete = true;
           if (flag.cont_exists = true) then
               do;
                   put edit("A continuous model exists, would you
 like to: ")(skip,a);
                   put skip;
                   put edit("1.  Generate a discrete model from t
he continuous system")(skip, a);
                   if (flag.discrete_exists = true) then
                       put edit ("2.  Modify your existing discret
e system")(skip,a);
                   else
                       put edit("2.  Enter a new discrete system")
(skip,a);
                   put edit("3.  Quit")(skip,a);
                   put skip;
                   put edit("Please choose one => ")(skip,a);
                   get list(c2);
                   range = 3;
                   call num_answer_ok (c2, range, choice2);

                   if (choice2 = 1) then do;
                       flag.discrete_exists = true;
                       flag.quantized_exists = false;
```

```
                    flag.control_law_valid = false;
                    flag.sim_valid = false;
                    if (flag.discrete_exists = true) then
                      free phi_matrix, lambda_matrix;
                    call build_discrete_matricies;
                    choice2 = 3;
                 end;
                 if (choice2 = 2) then do;
                    tau = 0.0;
                    goto Modify_dis;
                 end;
                 if (choice2 = 3) then
                    goto PARMS;
              end;
          else
             do;
                 put ecit("A continuous system does not exist,
would you like to")
                                                  (skip,a);
                 put ecit ("1.  Create a continuous system firs
t")(skip,a);
                 if (flag.discrete_exists = true) then
                    put edit ("2.  Modify the existing discrete
 system")(skip,a);
                 else
                    put edit ("2.  Enter a new discrete system"
)(skip,a);
                 put ecit ("3.  Quit")(skip,a);
                 put skip;
                 get list (c2);
                 range = 3;
                 call num_answer_ok (c2, range, choice2);

                 if (choice2 = 1) then do;
                    flag2.modify_discrete = false;
                    goto case(2);
                 end;

                 if (choice2 = 2) then do;
                    tau = 0.0;
                    goto Modify_dis;
                  end;
                 if (choice2 = 3) then
                    goto PARMS;
             end;

     Modify_dis:    if (flag.discrete_exists = true) then
                       do;
                          put edit ("Which of the following wou
ld you like to modify?")(skip,a);
                          put skip;
                          put edit("1.  Number of states")(skip
,a);
                          put edit("2.  Number of inputs")(skip
,a);
                          put edit("3.  Discrete system matrix"
)(skip,a);
                          put edit("4.  Discrete input matrix")
(skip,a);
                          put edit("5.  All of the above")(skip
,a);
                          put edit("6.  None of the above")(ski
p,a);
                          put skip;
                          put edit ("Please choose one => ")(sk
ip,a);
                          get list (c3);
                          range = 6;
```

```
                              call num_answer_ok (c3, range, choice
3);

                              if (choice3 ^= 6) then
                                 do;
                                    flag2.need_set = true;
                                    choice2 = choice3;
                                    goto case_2(choice2);
                                 end;
                              else
                                 do;
                                    if (flag2.need_set = true) then
                                       do;
                                          flag.discrete_exists = tr
ue;
                                          flag.quantized_exists = f
alse;
                                          flag.control_law_valid =
false;
                                          flag.sim_valid = false;
                                       end;
                                    goto PARMS;
                              end;
                           end;
                        else
                           do;
                                allocate phi_matrix, lambda_matrix;
                                flag2.need_set = true;
                                flag.discrete_exists = true;
                                choice2 = 5;
                                goto case_2(1);
                           end;

case(4):  /* Quantizec Parms */
          put skip;
          flag.own_quant_file_exists = false;
          if (flag.quantized_exists = true) then
             do;
                put edit ("A quantized system currently exists
")(skip,a);
                put edit ("Do you still wish to modify the qua
ntized system? => ")(skip,a);
                get list (answer);
                call yn_answer_ok (answer);
                if (answer = "n"| answer = "no") then
                   goto PARMS;
             end;
          if (flag.discrete_exists = true) then
             do;
                put edit("Would you like to generate a quantiz
ed system")(skip,a);
                put edit("           from the discrete system?
 => ")(skip,a);
                get list (answer);
                call yn_answer_ok (answer);
                if (answer = "y"| answer = "yes") then
                   do;
                      if (flag.quantized_exists = true) then
                         do;
                            free next_state_map;
                            free number_of_steps_s, number_of_
steps_i;
                            free voltage_lower_bound_s, voltag
e_upper_bound_s;
                            free voltage_lower_bound_i, voltag
e_upper_bound_i;
                            free offset_s, offset_i, quantum_s
```

```
tep_size_s, quantum_step_size_i;
                             free sat_edge, uncontrollable_cell
;
                           end;
                      allocate number_of_steps_s;
                      put edit ("Enter the number of quantizat
ion steps")(skip,a);
                      put skip;
                      do i = 1 to n;
                         put edit ("for state number", i, " =>
 ")(x(3), a, f(5), a);
                         get list (number_of_steps_s (i));
                      end;
                      allocate voltage_upper_bound_s,voltage_l
ower_bound_s;
                      put edit ("Enter the upper and lower vol
tage bounds (u, l)")(skip, a);
                      put skip;
                      do i = 1 to n;
                         put edit ("for state number", i, " =>
 ")(x(3), a, f(5), a);
                         get list (voltage_upper_bound_s (i),
voltage_lower_bound_s (i));
                      end;
                      allocate number_of_steps_i;
                      if ( p_real > 0 ) then
                         do;
                            put edit ("Enter the number of qua
ntization steps")(skip,a);
                            put skip;
                            do i = 1 to p;
                               put edit ("for input number", i
, " => ")(x(3), a, f(5), a);
                               get list (number_of_steps_i (i)
);
                            end;
                         end;
                      else
                         number_of_steps_i (1) = 1;

                      allocate voltage_upper_bound_i, voltage_
lower_bound_i;
                      if ( p_real > 0 ) then
                         do;
                            put edit ("Enter the upper and low
er voltage bounds (u, l)")(skip, a);
                            put skip;
                            do i = 1 to p;
                               put edit ("for input number", i
, " => ")(x(3), a, f(5), a);
                               get list (voltage_upper_bound_i
 (i), voltage_lower_bcund_i (i));
                            end;
                         end;
                      else
                         do;
                            voltage_upper_bound_i (1) = 0;
                            voltage_lower_bound_i (1) = 0;
                         end;

                      put edit ("Would you like the next state
 file built")(skip,a);
                      get list (answer);
                      if (answer = "y" | answer = "yes") then
do;
                         call build_misc_arrays;
                         call hcs_$initiate_count (working_dir
,
                                   job_name||".next_state", "", bit
_count,
                                     0, next_state_file_ptr, code);
                              call delete (job_name||".next_state",
 "-bf");
                              call hcs_$make_seg (working_dir, job_
name||
                               ".next_state", "", 01010b, next_stat
e_file_ptr, code);
                              call build_next_state_file;
                           end;
                        end;
                     else
                        do;
                           put edit ("Do you have a data file conta
ining the quantized")(skip,a);
                           put edit ("        system that you would
like to access? => ")(skip,a);
                           get list (answer);
                           call yn_answer_ok (answer);
                           if (answer = "y" | answer = "yes") then
                              flag.own_quant_file_exists = true;
                           else
                              goto PARMS;
                        end;
                  end;
            else
                  do;
                     put edit ("A discrete system does not exist.")
(skip,a);
                     put edit ("Would you like to create one? => "
)(skip,a);
                     get list (answer);
                     call yn_answer_ok (answer);
                     if (answer = "y" | answer = "yes") then
                        goto PARMS;
                     put edit ("Do you have a quantized system data
 file that you would like to access? => ")(skip,a);
                     get list (answer);
                     call yn_answer_ok (answer);
                     if (answer = "y" | answer = "yes") then
                        flag.own_quant_file_exists = true;
                     else
                        goto PARMS;
                  end;

            if (flag.own_quant_file_exists = true) then
                  do;

                    if (flag.quantized_exists = true) then
                       do;
                          free number_of_steps_s, number_of_steps_i
;
                          free voltage_lower_bound_s, voltage_upper
_bound_s;
                          free voltage_lower_bound_i, voltage_upper
_bound_i;
                          free offset_s, offset_i, quantum_step_siz
e_s, quantum_step_size_i;
                          free sat_edge, uncontrollable_cell;
                           free next_state_map;
                       end;
                     put edit ("Enter the name of the file to be re
ad in => ")(skip, a);
  /***/              get list (own_quant_data_title);
                     call hcs_$initiate_count (working_dir,
                        own_quant_data_title, "", bit_count, 0, ne
xt_state_file_ptr,
                         code);
                    if (next_state_file_ptr = null) then
                         do;
                              put edit ("The file", own_quant_data_tit
le, "does not exist")(skip,a);
                              put edit ("Try Again => ")(skip,a);
                              get list (own_quant_data_title);
                              call hcs_$initiate_count (working_dir,
                                   own_quant_data_title, "", bit_count,
0,
                                   next_state_file_ptr, code);
                              if (next_state_file_ptr = null) then
                                   goto PARMS;
                         end;

                         put edit("The following information is need
ed to suppliment the cuantization model: ")(skip,a);
                         put skip;
                         put edit ("the number of states => ")(skip,
a);
                          get list (n);
                         put edit ("the number of inputs => ")(skip,
a);
                          get list (p_real);
                         if (p_real > 0) then
                              p = p_real;
                         else
                              p = 1;
                              allocate number_of_steps_s;
                              put edit ("Enter the number of quantizat
ion steps")(skip,a);
                              put skip;
                              do i = 1 to n;
                                   put edit ("for state number", i, " =>
 ")(x(3), a, f(5), a);
                                   get list (number_of_steps_s (i));
                              end;
                              allocate voltage_upper_bound_s,voltage_l
ower_bound_s;
                              put edit ("Enter the upper and lower vol
tage bounds (u, l)")(skip, a);
                              put skip;
                              do i = 1 to n;
                                   put edit ("for state number", i, " =>
 ")(x(3), a, f(5), a);
                                   get list (voltage_upper_bound_s (i),
voltage_lower_bound_s (i));
                              end;
                              allocate number_of_steps_i;
                              if ( p_real > 0 ) then
                                   do;
                                        put edit ("Enter the number of qua
ntization steps")(skip,a);
                                        put skip;
                                        do i = 1 to p;
                                             put edit ("for input number", i
, " => ")(x(3), a, f(5), a);
                                             get list (number_of_steps_i (i)
);
                                        end;
                                   end;
                              else
                                   number_of_steps_i (1) = 1;

                              allocate voltage_upper_bound_i, voltage_
lower_bound_i;
                              if ( p_real > 0 ) then
                                   do;
```

```
                                                  put edit ("Enter the upper and low
er voltage bounds (u, l)")(skip, a);
                                                  put skip;
                                                  do i = 1 to p;
                                                      put edit ("for input number", i
, " => ")(x(3), a, f(5), a);
                                                      get list (voltage_upper_bound_i
 (i), voltage_lower_bound_i (i));
                                                  end;
                                              end;
                                         else
                                              do;
                                                  voltage_upper_bound_i (1) = 0;
                                                  voltage_lower_bound_i (1) = 0;
                                              end;
                                         call build_misc_arrays;


                                         allocate next_state_map;
                                         do i = 1 to num_state_combs;
                                              do j = 1 to num_input_combs;
                                                  next_state_map (i,j) = the_next_st
ate_mapping (i,j);
                                              end;
                                          end;

                         end;

                    call build_cont_reg_sat_edg_arrys;
                    call print_next_state_file;

                    flag.quantized_exists = true;
                    flag.control_law_valid = false;
                    flag.sim_valid = false;

                    goto PARMS;


case(5):            /*Quit*/
                    put skip;


build_discrete_matricies: procedure;

   dcl n fixed external;
   dcl p fixed external;

   dcl tau float external;
   dcl phi_matrix (1:n, 1:n) float controlled external;
   dcl lambda_matrix (1:n, 1:p) float controlled external;

   dcl i fixed;
   dcl j fixed;
   dcl matrix_dim fixed binary (35);
   dcl ind fixed binary (35);
   dcl ier fixed binary (35);

   dcl time float binary;
   dcl time_end float binary;
   dcl tol float binary;
   dcl c (1:24) float binary;
   dcl cont_state (10) float binary;
   dcl cont_input (1:p) float binary controlled external;
   dcl w (1:n, 1:9) float binary controlled;
   dcl temp_prime (1:n) float controlled external;

   dcl imsl$dverk entry (fixed binary (35), entry, float binary
, (*) float
```

```
                              binary, float binary, float binary, fi
xed binary (35),
                              (*) float binary, fixed binary (35), (
*, *) float
                              binary, fixed binary (35));

  dcl sysin file input;
  dcl sysprint file output;

  allocate cont_input, temp_prime, w, phi_matrix, lambda_matri
x;


  put edit ("Enter tau =>")(skip,x(4),a);
  get list (tau);
  do i = 1 to n;
    cont_state = 0;
    cont_state (i) = 1;
    matrix_dim = n;
    time = 0;
    time_end = tau;
    tol = .0001;
    ind = 1;
    call imsl$dverk (matrix_dim, equation_a, time, cont_state,
 time_end,
                           tol, ind, c, matrix_dim, w, ier);
    do j = 1 to n;
      phi_matrix (j, i) = cont_state (j);
    end;
  end;
  do i = 1 to p;
    cont_state = 0;
    cont_input = 0;
    cont_input (i) = 1;
    matrix_dim = n;
    time = 0;
    time_end = tau;
    tol = .0001;
    ind = 1;
    call imsl$dverk (matrix_dim, equation_b, time, cont_state,
 time_end,
                           tol, ind, c, matrix_dim, w, ier);
    do j = 1 to n;
      lambda_matrix (j, i) = cont_state (j);
    end;
  end;
  put edit ("PHI MATRIX = ")(skip(2), a);
  do i = 1 to n;
    put skip;
    do j = 1 to n;
      put list (phi_matrix (i, j));
    end;
  end;
  put edit ("LAMBDA MATRIX = ")(skip(2), a);
  do i = 1 to n;
    put skip;
    do j = 1 to p;
      put list (lambda_matrix (i, j));
    end;
  end;

  free cont_input, temp_prime, w;


  equation_a: procedure (matrix_dim, time, cont_state, cont_st
ate_prime);
```

```
     dcl matrix_dim fixed binary (35);
     dcl time float binary;
     dcl cont_state (10) float binary;
     dcl cont_state_prime (10) float binary;

     dcl n fixed external;

     dcl a_matrix (1:n, 1:n) float controlled external;

     dcl i fixed;
     dcl j fixed;

     do i = 1 to n;
       cont_state_prime (i) = 0;
       do j = 1 to n;
         cont_state_prime (i) = cont_state_prime (i) + (a_matri
x (i, j) *
                                         cont_state (j));
       end;
     end;

   end equation_a;


   equation_b: procedure (matrix_dim, time, cont_state, cont_st
ate_prime);

     dcl matrix_dim fixed binary (35);
     dcl time float binary;
     dcl cont_state (10) float binary;
     dcl cont_state_prime (10) float binary;
     dcl cont_input(1:p) float binary controlled external;

     dcl n fixed external;
     dcl p fixed external;
     dcl temp_prime (1:n) float controlled external;

     dcl a_matrix (1:n, 1:n) float controlled external;
     dcl b_matrix (1:n, 1:p) float controlled external;

     dcl i fixed;
     dcl j fixed;
     dcl k fixed;

     do i = 1 to n;
       temp_prime (i) = 0;
       do k = 1 to p;
         temp_prime(i) = temp_prime(i) + (b_matrix(i,k) * cont_
input(k));
       end;
     end;
     do i = 1 to n;
       cont_state_prime (i) = 0;
       do j = 1 to n;
         cont_state_prime (i) = cont_state_prime (i) + temp_pri
me (i) + (a_matrix (i, j) *
                                         cont_state (j));
       end;
     end;

   end equation_b;

end build_discrete_matricies;

   goto skip_the_entry;
build_misc_arrays: entry;
```

```
  allocate offset_s, cffset_i, quantum_step_size_s, quantum_st
ep_size_i;
  quantum_step_size_s = (voltage_upper_bound_s - voltage_lower
_bound_s) /
                                                             (nu
mber_of_steps_s);
  offset_s (1) = 1;
  do i = 2 to n;
    offset_s (i) = cffset_s (i-1) * number_of_steps_s (i-1);
  end;
  num_state_combs = offset_s (n) * number_of_steps_s (n);
  if ( p_real > 0 ) then
    do;
      quantum_step_size_i = (voltage_upper_bound_i - voltage_l
ower_bound_i) /
                                                            (num
ber_of_steps_i);
      offset_i (1) = 1;
      do i = 2 to p;
        offset_i (i) = offset_i (i-1) * number_of_steps_i (i-1
);
      end;
      num_input_combs = offset_i (p) * number_of_steps_i (p);
    end;
  else
    do;
      num_input_combs = 1;
      offset_i (1) = 1;
      quantum_step_size_i (1) = 1;
    end;

return;

  goto skip_the_entry;
build_cont_reg_sat_edg_arrys: entry;


  dcl found_sat_edge bit(1);
  dcl state_code1 fixed;
  dcl input_code1 fixed;
  dcl next_state_code1 fixed;
  dcl num_sat_edges fixed;

  allocate uncontrollable_cell, sat_edge;

  do state_code1 = 1 to num_state_combs;
    uncontrollable_cell (state_code1) = false;
  end;
  found_sat_edge = false;
  do state_code1 = 1 to num_state_combs;
    do input_code1 = 1 to num_input_combs;
      next_state_code1 = next_state_map (state_code1, input_co
de1);
      if ( next_state_code1 = 0 ) then
        do;
          sat_edge (state_code1, input_code1) = true;
          found_sat_edge = true;
        end;
      else
          sat_edge (state_code1, input_code1) = false;
    end;
  end;
  do while ( found_sat_edge = true );
    do state_code1 = 1 to num_state_combs;
      if ( uncontrollable_cell (state_code1) = false ) then
        do;
          num_sat_edges = 0;
          do input_coce1 = 1 to num_input_combs;
```

```
            if ( sat_edge (state_code1, input_code1) = true )
then
                num_sat_edges = num_sat_edges + 1;
          end;
          if ( num_sat_edges = num_input_combs ) then
            do;
              uncontrollable_cell (state_code1) = true;
            end;
        end;
      end;
      found_sat_edge = false;
      do state_code1 = 1 to num_state_combs;
        do input_code1 = 1 to num_input_combs;
          if ( sat_edge (state_code1, input_code1) = false ) the
n
            do;
              next_state_code1 = next_state_map (state_code1, in
put_code1);
              if ( uncontrollable_cell (next_state_code1) = true
 ) then
                do;
                  sat_edge (state_code1, input_code1) = true;
                  found_sat_edge = true;
                end;
            end;
        end;
      end;
    end;
    num_controllable_cells = 0;
    do state_code1 = 1 to num_state_combs;
      if ( uncontrollable_cell (state_code1) = false ) then
        num_controllable_cells = num_controllable_cells + 1;
    end;

return;

goto skip_the_entry;
build_next_state_file: entry;

    dcl next_state_code fixed;
    dcl state_code fixed;
    dcl input_code fixed;
    dcl state_code_temp fixed;
    dcl input_code_temp fixed;
    dcl dis_state (1:n) fixed controlled;
    dcl next_dis_state (1:n) fixed controlled;
    dcl dis_input (1:p) fixed controlled;

    dcl cont_state (1:n) float controlled;
    dcl cont_input (1:p) float controlled;
    dcl temp_1 (1:n) float controlled;
    dcl temp_2 (1:n) float controlled;

    dcl not_saturated bit(1);


    put skip;
     put edit ("Building next state file")(skip, x(4), a);
     allocate dis_state, dis_input, next_dis_state, cont_state,
cont_input,
            temp_1, temp_2;

       allocate next_state_map;

    do state_code = 1 to num_state_combs;
       do input_code = 1 to num_input_combs;
          state_code_temp = state_code;
          input_code_temp = input_code;

          call convert_$code_to_dis_state(state_code, dis_state)
;
          call convert_$code_to_dis_input (input_code, dis_input
);

          state_code = state_code_temp;
          input_code = input_code_temp;
          call add_entry;
       end;
    end;

     free dis_state, dis_input, next_dis_state, cont_state, cont
_input,
          temp_1, temp_2;


    add_entry: procedure;

      dcl i fixed;

      call convert_$dis_state_to_cont_state ((dis_state), cont_s
tate);
      do i = 1 to n;
        temp_1 (i) = 0;
        do j = 1 to n;
          temp_1 (i) = temp_1 (i) + phi_matrix (i,j) * cont_stat
e (j);
        end;
      end;
      call convert_$dis_input_to_cont_input ((dis_input), cont_i
nput);
      do i = 1 to n;
        temp_2 (i) = 0;
        do j = 1 to p;
          temp_2 (i) = temp_2 (i) + lambda_matrix (i, j) * cont_
input (j);
        end;
      end;
      temp_1 = temp_1 + temp_2;
      call convert_$cort_state_to_dis_state ((temp_1), next_dis_
state);

      not_saturated = true;
      do i = 1 to n while (not_saturated = true);
        if (next_dis_state (i) > number_of_steps_s (i) - 1 |
                                                     next_dis_sta
te (i) < 0) then
          not_saturated = false;
      end;
      if ( not_saturated = true ) then
        call convert_$dis_state_to_code ((next_dis_state), next_
state_code);
      else
        next_state_code = 0;
      next_state_map (state_code, input_code) = next_state_code;
      if (next_state_code ^= 0) then do;
      end;


    end add_entry;
    return;

skip_the_entry:  put skip;
end change_parameters;
```

```
print_it: procedure;

dcl job_name character(50) varying external;
dcl data_file file;
dcl 1 flag external,
        2 cont_exists bit(1),
        2 discrete_exists bit(1),
        2 quantized_exists bit(1),
        2 control_law_valid bit(1),
        2 sim_valid bit(1),
        2 own_quant_file_exists bit(1);

dcl done bit(1);
dcl answer character(3) varying;
dcl choice fixed;
dcl choice_c character(1);
dcl range fixed;

dcl num_answer_ok entry (character(1), fixed, fixed);
dcl yn_answer_ok entry (character(3) varying);
dcl print_next_state_file entry;
dcl build_tol_reg_and_cont_law$pr_cont_law entry;
dcl check_quantization_level entry;
dcl sysin file input;
dcl sysprint file output;

done = "0"b;
do while (done = "0"b);
   put skip(2);
   put edit ("Which of the following would you like printed?"
)(skip,a);
   put skip;
   range = 3;
   put edit ("1.  Status of the job")(skip,a);
   put edit ("2.  Data file for the job")(skip,a);
   if (flag.quantized_exists = "1"b) then
      do;
         range = 5;
         put edit ("3.  Quantized data file - next state file
 ")(skip,a);
         put edit ("4.  Quantization level check")(skip,a);
         if (flag.control_law_valid  = "1"b) then
            do;
               range = 6;
               put edit ("5.  Control law")(skip,a);
               put edit ("6.  None of the above")(skip,a);
            end;
          else
             put edit ("5.  None of the above")(skip,a);
        end;
   else do;
      range = 3;
      if (flag.control_law_valid = "1"b) then
         do;
            range = 4;
            put edit ("3.  Control law")(skip,a);
            put edit ("4.  None of the above")(skip,a);
         end;
      else
         put edit ("3.  None of the above")(skip,a);
   end;
   put skip;
   put edit ("Enter choice => ")(skip,a);
   get list (choice_c);
   call num_answer_ok (choice_c, range, choice);

   if (choice = 1) then
      do;
```

```
            put skip;
            if (flag.cont_exists = "1"b) then
               put edit ("     A continuous model of the system e
xists")(skip,a);
            if (flag.discrete_exists = "1"b) then
               put edit ("     A discrete model of the system exi
sts")(skip,a);
            if (flag.quantized_exists= "1"b) then
               put edit ("     A quantized model of the system ex
ists")(skip,a);
            if (flag.control_law_valid = "1"b) then
               put edit ("     A control law exists for the  job"
)(skip,a);
            if (flag.sim_valid = "1"b) then
               put edit ("     The system has been successfully s
imulated")(skip,a);
         end;

        if (choice = 2) then call display_job_file;
      if (choice = 3) then
         do;
            if (range = 3) then done = "1"b;
            if (range = 4) then call build_tol_reg_and_cont_law$
pr_cont_law;
            if (range = 5 | range = 6) then call print_next_stat
e_file;
         end;

      if (choice = 4) then
         do;
            if (range = 4) then done = "1"b;

            if (range = 5|range = 6) then call check_quantizatio
n_level;
         end;

      if(choice = 5) then
         do;
            if (range = 5) then done = "1"b;
            if (range = 6) then call build_tol_reg_and_cont_law$
pr_cont_law;
         end;

      if (choice = 6) then done = "1"b;

    end; /* while */
goto endit;

 display_job_file: entry;

  put edit ("OOPs I haven't written this one yet'")(skip,a);

 return;

endit:   end print_it;
```

```
print_next_state_file: procedure;

   dcl num_state_combs fixed external;
   dcl num_input_combs fixed external;

   dcl next_state_file_ptr pointer external;

   dcl next_state_map (1:num_state_combs, 1:num_input_combs) fi
xed controlled external;

   dcl the_next_state_mapping (1:num_state_combs, 1:num_input_c
ombs) fixed binary (18)
                        unsigned based (next_state_file_ptr);

   dcl yn_answer_ok entry (character(3) varying);
   dcl sysin file input;
   dcl sysprint file cutput;

   dcl i fixed;
   dcl j fixed;
   dcl count fixed;

   dcl answer character (3) varying;

   put edit ("Would ycu like the next state file printed? => ")
(skip,a);
   get list (answer);
   call yn_answer_ok(answer);
   if (answer = "yes" | answer = "y") then
     do;
       count = 0;
       do i = 1 to num_state_combs;
         count = count + 1;
         if (count = 11) then
           do;
              count = 1;
              put edit ("More? => ")(skip,a);
              get list (answer);
              call yn_answer_ok (answer);
              put skip;
              if (answer = "n" | answer = "no") then goto nomor
e;
            end;
         put edit (i)(skip,f(4));;
         do j = 1 to rum_input_combs;
           put edit (next_state_map (i, j))(x(2), f(7));
         end;
       end;
     end;

nomore:  put skip;
end print_next_state_file;
```

```
check_quantization_level: procedure;

   dcl n fixed external;
   dcl p fixed external;
   dcl p_real fixed external;
   dcl num_state_combs fixed external;
   dcl num_input_combs fixed external;
   dcl offset_s (1:n) fixed controlled external;
   dcl offset_i (1:p) fixed controlled external;
   dcl number_of_steps_s (1:n) fixed controlled external;
   dcl number_of_steps_i (1:p) fixed controlled external;

   dcl voltage_lower_bound_s (1:n) float controlled external;
   dcl voltage_lower_bound_i (1:p) float controlled external;
   dcl quantum_step_size_s (1:n) float controlled external;
   dcl quantum_step_size_i (1:p) float controlled external;
   dcl num_controllable_cells fixed external;

   dcl next_state_file_ptr pointer external;

   dcl state (1:n) fixed controlled;
   dcl input (1:p) fixed controlled;
   dcl next_state (1:n) fixed controlled;

   dcl next_state_map (1:num_state_combs, 1:num_input_combs) fi
xed controlled external;

   dcl the_next_state_mapping (1:num_state_combs, 1:num_input_c
ombs) fixed binary (18)
                          unsigned based (next_state_file_ptr);

   dcl answer character(3) varying;

   dcl convert_$dis_state_to_code entry ((*) fixed, fixed);
   dcl convert_$dis_input_to_code entry ((*) fixed, fixed);
   dcl convert_$code_to_dis_state entry (fixed, (*) fixed);
   dcl convert_$code_to_dis_input entry (fixed, (*) fixed);
   dcl yn_answer_ok entry (character(3) varying);

   dcl sysin file input;
   dcl sysprint file output;

   put edit ("Would you like to check the quantization level ?
")(skip,a);
   get list (answer);
   call yn_answer_ok (answer);
   if (answer = "yes" | answer = "y") then
     do;
       put edit ("Number of controllable cells = ")(skip,a);
       put list (num_controllable_cells);
       put edit ("      Total number of cells  =   ")(skip,a);
       put list (num_state_combs);
       put skip;
       allocate state, input, next_state;
       call find_zero_input_dists;
       if ( p_real > 0 ) then call find_zero_state_dists;
       free state, input, next_state;
       call print_sat_edge_array;
       if (num_controllable_cells ^= num_state_combs) then do;
          call print_controllable_cells;
          call print_uncontrollable_cells;
       end;
     end;


   find_zero_input_dists: procedure;

     dcl i fixed;
```

```
      dcl s_level fixed;
      dcl state_code fixed;
      dcl state_code_temp fixed;
      dcl next_state_code fixed;
      dcl zero_input_code fixed;
      dcl sum_num_steps fixed;
      dcl max_num_steps fixed;
      dcl num_tot fixed;
      dcl zero_input (1:p) fixed controlled;
      dcl num_dir (1:n) fixed controlled;
      dcl cells_moved_tot (0:sum_num_steps) fixed controlled;
      dcl cells_moved_dir (1:n, 0:max_num_steps) fixed controlle
d;

        max_num_steps = number_of_steps_s (1);
        sum_num_steps = 0;
        do i = 1 to n;
          max_num_steps = max (max_num_steps, number_of_steps_s
(i));
          sum_num_steps = sum_num_steps + number_of_steps_s (i);
        end;

        allocate zero_input, num_dir, cells_moved_tot, cells_mov
ed_dir;

        zero_input = floor ( -voltage_lower_bound_i / quantum_st
ep_size_i);
        call convert_$dis_input_to_code ((zero_input), zero_inpu
t_code);
        do state_code = 1 to num_state_combs;
         state_code_temp = state_code;
         call convert_$code_to_dis_state (state_code, state);
         state_code = state_code_temp;
         call compute_zero_input_dists;
        end;
        call print_zero_input_dists;

        free zero_input, num_dir, cells_moved_tot, cells_moved_d
ir;



      compute_zero_input_dists: procedure;

        if ( next_state_map (state_code, zero_input_code) > 0 )
then
          do;
            next_state_code = next_state_map (state_code, zero_i
nput_code);
            call convert_$code_to_dis_state ((next_state_code),
next_state);
            num_dir = abs (state - next_state);
            num_tot = 0;
            do i = 1 to n;
              cells_moved_dir (i,num_dir(i))= cells_moved_dir (i
,num_dir(i)) + 1;
              num_tot = num_tot + num_dir (i);
            end;
            cells_moved_tot (num_tot) = cells_moved_tot (num_tot
) + 1;
          end;

      end compute_zero_input_dists;

      print_zero_input_dists: procedure;

        dcl i fixed;
        dcl j fixed;
        dcl max_cells_moved fixed;
        dcl most_cells_moved (1:max_num_steps) fixed controlled;

        dcl zero_cells_moved bit(1);
        dcl true bit(1) initial ("1"b);
        dcl false bit(1) initial ("0"b);

        allocate most_cells_moved;

        do i = 1 to n;
          zero_cells_moved = true;
          do j = max_num_steps by -1 to 0 while (zero_cells_move
d = true);
            if (cells_moved_dir (i, j) > 0 ) then
              do;
                zero_cells_moved = false;
                most_cells_moved (i) = j;
              end;
          end;
        end;
        max_cells_moved = most_cells_moved (1);
        do i = 1 to n;
          max_cells_moved = max (max_cells_moved, most_cells_mov
ed (i));
        end;
        put edit ("Number of cells moved in each direction")(ski
p,x(6),a);
        put edit ("Dir")(skip,a,x(3));
        do i = 0 to max_cells_moved;
          put edit (i)(f(4));
        end;
        do i = 1 to n;
          put edit (i)(skip,f(4),x(3));
          do j = 0 to max_cells_moved;
            put edit (cells_moved_dir (i,j)) (f(4));
          end;
        end;
        zero_cells_moved = true;
        do i = sum_num_steps by -1 to 0 while (zero_cells_moved
= true);
          if ( cells_moved_tot (i) > 0 ) then
            do;
              zero_cells_moved = false;
              max_cells_moved = i;
            end;
        end;
        put skip;
        put edit ("Number of cells moved total")(skip,x(6),a);
        put edit ("num")(skip,a);
        do i = 0 to max_cells_moved;
          put edit (i)(f(4));
        end;
        put edit ("   ")(skip,a);
        do i = 0 to max_cells_moved;
          put edit (cells_moved_tot (i))(f(4));
        end;

        free most_cells_moved;

      end print_zero_input_dists;

    end find_zero_input_dists;

    find_zero_state_dists: procedure;

      dcl i fixed;
      dcl j fixed;
```

```
     dcl input_code fixed;
     dcl next_state_code fixed;
     dcl zero_state_code fixed;
     dcl zero_state (1:n) fixed controlled;
     dcl zero_input (1:p) fixed controlled;
     dcl num_input_steps (1:p) fixed controlled;
     dcl cells_moved_tot_abs (1:p) fixed controlled;
     dcl cells_moved_dir_abs (1:p, 1:n) fixed controlled;

     dcl cells_moved_tot_avg (1:p) float controlled;
     dcl cells_moved_dir_avg (1:p, 1:n) float controlled;

     dcl input_status (1:p) character(9) controlled;

     dcl found_not_sat bit(1);
     dcl true bit(1) initial ("1"b);
     dcl false bit(1) initial ("0"b);

     allocate zero_state, zero_input, num_input_steps, cells_mo
ved_tot_abs,
              cells_mcved_tot_avg,  cells_moved_dir_abs, cells_
moved_dir_avg,
              input_status;

     zero_state = floor (-voltage_lower_bound_s / quantum_step_
size_s);
     zero_input = floor (-voltage_lower_bound_i / quantum_step_
size_i);
     call convert_$dis_state_to_code ((zero_state), zero_state_
code);

     call compute_zero_state_dists;
     call print_zero_state_dists;

     free zero_state, zero_input, num_input_steps, cells_moved_
tot_abs,
          cells_moved_tot_avg,  cells_moved_dir_abs, cells_move
d_dir_avg,
          input_status;


     compute_zero_state_dists: procedure;

        do i = 1 to p;
           input = zero_input;
           found_not_sat = false;
           input (i) = 0;
           do while (input (i) < zero_input (i)   &   found_not_s
at = false );
              call convert_$dis_input_to_code ((input), input_code
);
              if ( next_state_map (zero_state_code, input_code) >
0 ) then
                 found_not_sat = true;
              else
                 input (i) = input (i) + 1;
           end;
           if ( found_not_sat = true ) then
              do;
                 if ( input (i) = 0 ) then input_status (i) = "max
unsat";
                 else input_status (i) = "max satur";
                 num_input_steps (i) = zero_input (i) - input (i);
                 next_state_code = next_state_map (zero_state_code,
 input_code);
                 call convert_$code_to_dis_state ((next_state_code)
, next_state);
                 cells_moved_dir_abs (i,*) = abs (next_state - zero
_state);
              cells_moved_dir_avg (i,*) = cells_moved_dir_abs (i
,*) /
                                                                  num
_input_steps (i);
              cells_moved_tot_abs (i) = 0;
              do j = 1 to n;
                cells_moved_tot_abs (i) = cells_moved_tot_abs (i
) +
                                                            cells_move
d_dir_abs (i, j);
              end;
              cells_moved_tot_avg (i) = cells_moved_tot_abs (i)
/
                                                                  num
_input_steps (i);
            end;
          else
            input_status (i) = "all satur";
        end;

      end compute_zero_state_dists;

      print_zero_state_dists: procedure;

        put skip;
        put edit ("Input    Input     Num   Total Cells        Ce
lls Moved")(skip,a);
        put edit ("         Status   Input     Moved
    in")(skip,a);

        put edit ("                  Steps   Abs   Avg          D
ir   Abs    Avg")(skip,a);
        do i = 1 to p;
          put skip;
          if ( input_status (i) = "all satur" ) then
            do;
              put edit (i, input_status (i))(x(4),f(4),x(3),a(9)
);
            end;
          else
            do;
              put edit (i, input_status (i), num_input_steps (i)
)
                             (f(4), x(3), a(9), x(3), f(4));
              put edit (cells_moved_tot_abs (i), cells_moved_tot
_avg (i), " ")
                             (x(4), f(4), x(2), f(6,2), a);
              do j = 1 to n;
                put edit (j, cells_moved_dir_abs (i,j), cells_mo
ved_dir_avg (i,j), " ")
                (skip, x(42), f(4), x(2), f(4), x(5), f(6,2), a)
;
              end;
            end;
        end;
        put skip;

      end print_zero_state_dists;

    end find_zero_state_dists;


  print_sat_edge_array: procedure;

    dcl num_state_combs fixed external;
    dcl num_input_combs fixed external;
```

```
   dcl sat_edge (1:num_state_combs, 1:num_input_combs) bit (1)
controlled
                 external;

   dcl yn_answer_ok entry (character(3) varying);
   dcl sysin file input;
   dcl sysprint file cutput;

   dcl i fixed;
   dcl j fixed;
   dcl count fixed;

   dcl answer character (3) varying;

   put edit ("Would you like the saturated edge array printed?
")(skip,a);
   get list (answer);
   call yn_answer_ok (answer);
   if (answer = "yes" | answer = "y") then
     do;
       count = 0;
       do i = 1 to num_state_combs;
         count = count + 1;
         if (count = 11) then
           do;
              count = 1;
              put edit ("More? => ")(skip,a);
              get list (answer);
              call yn_answer_ok (answer);
              put skip;
              if (answer = "n" | answer = "no") then goto nomor
e;
           end;
         put edit (i)(skip,f(4));;
         do j = 1 to num_input_combs;
           if ( sat_edge (i, j) = "0"b ) then put edit ("    F"
)(a);
           else put edit ("    T")(a);
         end;
       end;
     end;
nomore:  put skip;

end print_sat_edge_array;

print_controllable_cells: procedure;

   dcl num_state_combs fixed external;

   dcl uncontrollable_cell (1:num_state_combs) bit(1) controlle
d external;

   dcl i fixed;
   dcl count fixed;
   dcl answer character(3) varying;
   dcl sysin file input;
   dcl sysprint file output;
   dcl yn_answer_ok entry (character(3) varying);

   put edit ("Would you like the controllable cells listed ? ")
(skip,a);
   get list (answer);
   call yn_answer_ok (answer);
   if ( answer = "yes" | answer = "y" ) then
     do;
       count = 0;
       put edit ("Controllable Cells")(skip,a);
       do i = 1 to num_state_combs;
```

```
            if ( uncontrollable_cell (i) = "0"b ) then
              do;
                 put skip list (i);
                 count = count + 1;
                 if (count = 10) then
                    do;
                      count = 1;
                      put edit ("More? => ")(skip,a);
                      get list (answer);
                      call yn_answer_ok (answer);
                      put skip;
                      if (answer = "n" | answer = "no") then goto
nomore;
                  end;
              end;
          end;
        end;
nomore:  put skip;

end print_controllable_cells;

print_uncontrollable_cells: procedure;

  dcl num_state_combs fixed external;

  dcl uncontrollable_cell (1:num_state_combs) bit(1) controlle
d external;

  dcl count fixed;
  dcl i fixed;
  dcl answer character(3) varying;
  dcl sysin file input;
  dcl sysprint file output;
  dcl yn_answer_ok entry (character(3) varying);

  put edit ("Would you like the uncontrollable cells listed ?
")(skip,a);
  get list (answer);
  call yn_answer_ok (answer);
  if ( answer = "yes" | answer = "y" ) then
    do;
      count = 0;
      put edit ("Uncontrollable Cells")(skip,a);
      do i = 1 to num_state_combs;
        if ( uncontrollable_cell (i) = "1"b ) then
          do;
             put skip list (i);
             count = count + 1;
             if (count = 11) then
               do;
                 count = 1;
                 put edit ("More? => ")(skip,a);
                 get list (answer);
                 call yn_answer_ok (answer);
                 put skip;
                 if (answer = "n" | answer = "no") then goto
nomore;
               end;
           end;
      end;
    end;
nomore:  put skip;

end print_uncontrollable_cells;

end check_quantization_level;
```

```
build_tol_reg_and_cont_law: procedure;

   dcl num_state_combs fixed external;
   dcl 1 flag external,
          2 cont_exists bit(1),
          2 discrete_exists bit(1),
          2 quantized_exists bit(1),
          2 control_law_valid bit(1),
          2 sim_valid bit(1),
          2 own_quant_file_exists bit(1);

   dcl cost_function_code fixed external;
   dcl cell_status_index fixed;
   dcl center_cell_tolerance fixed;
   dcl edge_cell_tolerance fixed;
   dcl cell_status (1:num_state_combs) fixed controlled;
   dcl center_dist (1:num_state_combs) fixed controlled;
   dcl found_all_loops bit(1);
   dcl control_law_file_ptr pointer external;
   dcl control_law (1:num_state_combs) fixed based (control_law
_file_ptr);
   dcl i fixed;

   dcl answer character(3) varying;

   dcl min_time_opt_cont_law entry ( (*) fixed, fixed);
   dcl yn_answer_ok entry (character (3) varying);
   dcl sysin file input;
   dcl sysprint file output;


   allocate cell_status, center_dist;

   if (flag.control_law_valid = "1"b) then
      put edit ("Would you like to rebuild the control law file
? ")(skip (2),a);
   else
      put edit ("Would you like to build the control law file?
")(skip(2), a);
   get list (answer);
   call yn_answer_ok (answer);
   if ( answer =  "y" | answer = "yes" ) then
   do;
     call build_cost_function;
     if (cost_function_code = 5 | cost_function_code = 6) then
do;
       goto dont_build;
     end;

     call get_tolerances (center_cell_tolerance, edge_cell_tole
rance);
     call initialize_cell_status_array (cell_status, edge_cell_
tolerance);
     call initialize_center_dist_array (center_dist, center_cel
l_tolerance);
     call find_root_cells (cell_status, center_dist, center_cel
l_tolerance,
                                    cell_status_index);
     call open_control_law_file;
     call find_loops_and_cont_law (cell_status, center_dist, ce
ll_status_index, center_cell_tolerance, found_all_loops);
     if (found_all_loops = "0"b) then goto dont_build;

       call build_optimal_control_law (cell_status, cell_status
_index);

     flag.sim_valid = "0"b;
     flag.control_law_valid = "1"b;
```

```
    call print_cell_status ((cell_status));

   dont_build:
    if (cost_function_code ^= 6 & flag.control_law_valid = "1"
b) then
       call print_control_law;
  end;

  free cell_status, center_dist;


  build_cost_function: procedure;

  dcl 1 flag external,
          2 cont_exists bit(1),
          2 discrete_exists bit(1),
          2 quantized_exists bit(1),
          2 control_law_valid bit(1),
          2 sim_valid bit(1),
          2 own_quant_file_exists bit(1);

     dcl true bit(1) initial ("1"b);
     dcl false bit(1) initial ("0"b);

     dcl working_dir character(168) external;
     dcl n fixed external;
     dcl p fixed external;
     dcl p_real fixed external;
     dcl cost_function_code_char character (1);
     dcl cost_function_code fixed external;
     dcl control_law_file_ptr pointer external;

     dcl state_cost_matrix (1:n) float controlled external;
     dcl input_cost_matrix (1:p) float controlled external;

     dcl range fixed;
     dcl i fixed;
     dcl bit_count fixed bin(24);
     dcl code fixed bin(35);
     dcl own_control_law_file character(70) external;

     dcl control_law (1:num_state_combs) fixed based (control_l
aw_file_ptr);

     dcl null builtin;
     dcl num_answer_ok entry  (character(1), fixed, fixed);
     dcl hcs_$initiate_count entry (char(*), char(*), char(*),
                    fixed bin(24), fixed bin(2), ptr, fixed bin
(35));
     dcl sysin file input;
     dcl sysprint file output;

   if (flag.control_law_valid = "1"b) then do;
      flag.control_law_valid = "0"b;
      if (cost_function_code = 2) then do;
         free state_cost_matrix;
         free input_cost_matrix;
      end;
      if (cost_function_code = 3) then free input_cost_matrix;
   end;

     put edit ("Which type of cost function would you like to u
se ?")(skip,a);
     put skip;
     put edit ("1) Minimum Time")(skip, x(4), a);
     put edit ("2) Quadratic")(skip, x(4), a);
     put edit ("3) Minimum Control Effort")(skip, x(4), a);
     put edit ("4) Custom Cost Function (use procedure custom_c
ost_function.pl1")
                (skip, x(4), a);
    put edit ("5) None - Would like to access a control law fi
le")(skip, x(4), a);
    put edit ("6) None of the above")(skip, x(4), a);

    put edit ("    ")(skip,a);
    put edit ("Please choose one =>  ")(skip,a);
    get list (cost_function_code_char);
    range = 6;
    call num_answer_ok (cost_function_code_char, range, cost_f
unction_code);
    goto case (cost_function_code);

  case (1):goto done;

  case (2): allocate state_cost_matrix, input_cost_matrix;
            put skip;
            put edit ("Enter the value of...")(skip,a);
            put skip;
            do i = 1 to n;
              put edit ("state cost matrix (",i,",",i,") => ")
                        (x(4), a, f(3), a, f(3), a);
              get list (state_cost_matrix (i));
            end;
            put skip;
            if ( p_real > 0 ) then
              do;
                do i = 1 to p;
                  put edit ("input cost matrix (",i,",",i,") =
> ")
                            (x(4), a, f(3), a, f(3), a);
                  get list (input_cost_matrix (i));
                end;
              end;
            else
              input_cost_matrix (1) = 0;
            goto done;

  case (3): if ( p_real > 0 ) then
              do;
                 allocate input_cost_matrix;
                 if ( p_real > 1 ) then
                   do;
                     put skip;
                     put edit ("Enter scaling factor for...")(
skip,a);
                     put skip;
                     do i = 1 to p;
                       put edit ("Input (",i,") => ")(a,f(3),a
);
                       get list (input_cost_matrix (i));
                     end;
                   end;
                 else
                   input_cost_matrix (1) = 1;
                 end;
             else
               put edit ("System has no input.  This control m
akes no sense")
                         (skip, x(4), a);
            goto done;

  case (4): goto done;

  case (5):  put edit ("Enter the name of the control file to
be read in =>  ")(skip,a);
  get list (own_control_law_file);
```

```
   call hcs_$initiate_count (working_dir, own_control_law_file,
 "",
                              bit_count, 0, control_law_file_ptr, c
ode);
  if (control_law_file_ptr = null) then
    do;
      put edit ("The file", own_control_law_file, "does not ex
ist")(skip,a);
      put edit ("Try Again => ")(skip,a);
      get list (own_control_law_file);
      call hcs_$initiate_count (working_dir, own_control_law_f
ile,
                            "",bit_count, 0, control_law_file_ptr,
code);
      if (control_law_file_ptr = null) then goto done;
      else do;
        flag.control_law_valid = true;
      end;
    end;
    else do;
        flag.control_law_valid = true;
    end;

  case (6): goto done;


  done:     put skip;

  end build_cost_function;


  get_tolerances: procedure (center_cell_tolerance, edge_cell_
tolerance);

    dcl center_cell_tolerance fixed;
    dcl edge_cell_tolerance fixed;

    dcl sysin file input;
    dcl sysprint file output;

    put edit ("Enter the center cell tolerance => ")(skip, a);
    get list (center_cell_tolerance);
    put edit ("Enter the edge cell tolerance => ")(skip, a);
    get list (edge_cell_tolerance);

  end get_tolerances;


  initialize_cell_status_array: procedure (cell_status, edge_c
ell_tolerance);

    dcl cell_status (*) fixed;
    dcl edge_cell_tolerance fixed;

    dcl n fixed external;
    dcl num_state_combs fixed external;
    dcl number_of_steps_s (1:n) fixed controlled external;

    dcl uncontrollable_cell (1:num_state_combs) bit(1) control
led external;

    dcl state_code fixed;
    dcl recurse_level fixed;
    dcl dis_state (1:n) fixed controlled;

    dcl true bit(1) initial ("1"b);
    dcl convert_$dis_state_to_code entry ((*) fixed, fixed);


    allocate dis_state;

    cell_status = 2;
    recurse_level = n;
    call clear_all_but_edges;
    do state_code = 1 to num_state_combs;
      if ( uncontrollable_cell ( state_code) = true ) then
        cell_status (state_code) = 1;
    end;

    free dis_state;

    clear_all_but_edges: procedure recursive;

      if ( recurse_level <= 0 ) then
        do;
          call convert_$dis_state_to_code ((dis_state), state_
code);
          cell_status (state_code) = 0;
        end;
      else
        do;
          do dis_state (recurse_level) = edge_cell_tolerance t
o
               (number_of_steps_s (recurse_level) - (edge_cell_
tolerance + 1));
            recurse_level = recurse_level - 1;
            call clear_all_but_edges;
          end;
        end;
      recurse_level = recurse_level + 1;

    end clear_all_but_edges;

  end initialize_cell_status_array;


  initialize_center_dist_array: procedure (center_dist, center
_cell_tolerance);

    dcl center_dist (*) fixed;
    dcl center_cell_tolerance fixed;

    dcl n fixed external;

    dcl i fixed;
    dcl recurse_level fixed;
    dcl center_cell_tcl_index fixed;
    dcl state_code fixed;
    dcl dis_state (1:n) fixed controlled;
    dcl zero_dis_state (1:n) fixed controlled;
    dcl l_bound (1:n) fixed controlled;
    dcl u_bound (1:n) fixed controlled;

    dcl cont_state (1:n) float controlled;

    dcl convert_$cont_state_to_dis_state entry ((*) float, (*)
 fixed);
    dcl convert_$dis_state_to_code entry ((*) fixed, fixed);


    allocate dis_state, zero_dis_state, cont_state, l_bound, u
_bound;

    center_dist = 0;
    cont_state = 0;
    call convert_$cont_state_to_dis_state ((cont_state), zero_
dis_state);
    do center_cell_tol_index = center_cell_tolerance by -1 to
0;
      do i = 1 to n;
        l_bound (i) = zero_dis_state (i) - center_cell_tol_ind
ex;
        u_bound (i) = zero_dis_state (i) + center_cell_tol_ind
ex;
      end;
      recurse_level = n;
      call add_cent_tol_code;
    end;

    free dis_state, zero_dis_state, cont_state, l_bound, u_bou
nd;


    add_cent_tol_code: procedure recursive;

      if ( recurse_level <= 0 ) then
        do;
          call convert_$dis_state_to_code ((dis_state), state_
code);
          center_dist (state_code) = center_cell_tol_index;
        end;
      else
        do;
          do dis_state (recurse_level) = l_bound (recurse_leve
l) to
                                                        u_bound
 (recurse_level);
            recurse_level = recurse_level - 1;
            call add_cent_tol_code;
          end;
        end;
      recurse_level = recurse_level + 1;

    end add_cent_tol_code;

  end initialize_center_dist_array;


  find_root_cells: procedure (cell_status, center_dist, center
_cell_tolerance,
                                      cell_status_index);

    dcl cell_status (*) fixed;
    dcl center_dist (*) fixed;
    dcl center_cell_tolerance fixed;
    dcl cell_status_index fixed;

    dcl n fixed external;
    dcl num_state_comts fixed external;

    dcl i fixed;
    dcl max_num_cells fixed;
    dcl num_cells_reachable_to fixed;
    dcl best_root_code fixed;
    dcl zero_state_code fixed;
    dcl center_cell_tol_index fixed;
    dcl dis_state (1:n) fixed controlled;

    dcl cont_state (1:n) float controlled;

    dcl unmarked_cells bit(1);
```

```
     dcl possible_root bit(1);
     dcl true bit(1) initial ("1"b);
     dcl false bit(1) initial ("0"b);

     dcl convert_$cont_state_to_dis_state entry ((*) float, (*)
  fixed);
     dcl convert_$dis_state_to_code entry ((*) fixed, fixed);


     allocate dis_state, cont_state;

     cont_state = 0;
     call convert_$cont_state_to_dis_state ((cont_state), dis_s
tate);
     call convert_$dis_state_to_code ((dis_state), zero_state_c
ode);
     cell_status_index = 3;
     cell_status (zero_state_code) = cell_status_index;
     call add_cells_reachable_to (zero_state_code, cell_status)
;
     call check_for_unmarked_cells (unmarked_cells, cell_status
);
     do center_cell_tol_index = 1 to center_cell_tolerance
                                                   while (unmark
ed_cells = true);
       call check_for_possible_root (possible_root, center_cell
_tol_index,
                                                       cell_stat
us, center_dist);
       do while ( possible_root = true );
         max_num_cells = 0;
         do i = 1 to num_state_combs;
           if (center_dist (i)= center_cell_tol_index & cell_st
atus (i)= 0) then
             do;
               call find_num_cells_reachable_to ((i), (cell_sta
tus),
                                                       num_cells
_reachable_to);
               if ( num_cells_reachable_to > max_num_cells ) th
en
                 do;
                   best_root_code = i;
                   max_num_cells = num_cells_reachable_to;
                 end;
             end;
         end;
         cell_status_index = cell_status_index + 2;
         cell_status (best_root_code) = cell_status_index;
         call add_cells_reachable_to (best_root_code, cell_stat
us);
         call check_for_possible_root (possible_root, center_ce
ll_tol_index,
                                                       cell_stat
us, center_dist);
       end;
       call check_for_unmarked_cells (unmarked_cells, cell_stat
us);
     end;
     call check_for_unmarked_cells (unmarked_cells, cell_status
);
     if ( unmarked_cells = false ) then
       do;
         put edit ("Tree sucessfully completed")(skip,x(8),a);
       end;
     else
       do;
         put edit ("Trees Unsucessfully completed")(skip,x(8),a
);
         end;

      free dis_state, ccnt_state;


      check_for_unmarkec_cells: procedure (unmarked_cells, cell_
status);

         dcl unmarked_cells bit(1);
         dcl cell_status (*) fixed;

         dcl num_state_ccmbs fixed external;

         dcl i fixed;

         dcl true bit(1) initial ("1"b);
         dcl false bit(1) initial ("0"b);


         unmarked_cells = false;
         do i = 1 to num_state_combs while (unmarked_cells = fals
e);
           if ( cell_status (i) = 0 ) then
             unmarked_cells = true;
         end;

      end check_for_unmarked_cells;


      check_for_possible_root: procedure (possible_root, pos_roo
t_code,
                                                cell_status, center_di
st);

         dcl pos_root_ccce fixed;
         dcl possible_rcct bit(1);
         dcl cell_status (*) fixed;
         dcl center_dist (*) fixed;

         dcl num_state_ccmbs fixed external;

         dcl i fixed;

         dcl true bit(1) initial ("1"b);
         dcl false bit(1) initial ("0"b);


         possible_root = false;
         do i = 1 to num_state_combs while (possible_root = false
);
           if ( center_dist (i) = pos_root_code & cell_status (i)
 = 0 ) then
             possible_rcct = true;
         end;

      end check_for_possible_root;


      find_num_cells_reachable_to: procedure (cell_code, temp_ce
ll_stat, num_cells);

         dcl cell_code fixed;
         dcl temp_cell_stat (*) fixed;
         dcl num_cells fixed;
```

```
        dcl num_state_ccmbs fixed external;

        dcl i fixed;


        temp_cell_stat (cell_code) = 99;
        call add_cells_reachable_to (cell_code, temp_cell_stat);
        num_cells = 0;
        do i = 1 to num_state_combs;
          if ( temp_cell_stat (i) = 100 ) then num_cells = num_c
ells + 1;
        end;

     end find_num_cells_reachable_to;


     add_cells_reachable_to: procedure (root_code, cell_status)
;

        dcl root_code fixed;
        dcl cell_status (*) fixed;

        dcl num_state_combs fixed external;
        dcl num_input_ccmbs fixed external;

        dcl sat_edge (1:num_state_combs, 1:num_input_combs) bit(
1) controlled
                          external;

        dcl next_state_file_ptr pointer external;

        dcl state_code fixed;
        dcl input_code fixed;
        dcl next_state_code fixed;
        dcl root_status_code fixed;

   dcl next_state_map (1:num_state_combs, 1:num_input_combs) fi
xed controlled external;

        dcl the_next_state_mapping (1:num_state_combs, 1:num_inp
ut_combs) fixed binary(18)
                                unsigned based (next_state_file_ptr);

        dcl found_reachable_to_cell bit(1);
        dcl found_good_input bit(1);
        dcl true bit(1) initial ("1"b);
        dcl false bit(1) initial ("0"b);


        root_status_code = cell_status (root_code);
        found_reachable_to_cell = true;
        do while ( found_reachable_to_cell = true );
          found_reachable_to_cell = false;
          do state_code = 1 to num_state_combs;
            if ( cell_status (state_code)= 0 | cell_status (stat
e_code)= 2 ) then
              do;
                found_good_input = false;
                do input_code= 1 to num_input_combs while (found
_good_input= false);
                  if ( sat_edge (state_code, input_code) = false
 ) then
                    do;
                      next_state_code = next_state_map (state_co
de, input_code);
                      if ( cell_status (next_state_code) = root_
status_code |
                              cell_status (next_state_code) = root_
status_code + 1 )
                              then found_good_input = true;
                    end;
                end;
                if ( found_good_input = true ) then
                  do;
                    cell_status (state_code) = root_status_code
+ 1;
                    found_reachable_to_cell = true;
                  end;
              end;
          end;
        end;

      end add_cells_reachable_to;

    end find_root_cells;


    open_control_law_file: procedure;

      dcl num_state_combs fixed external;

      dcl job_name character (50) varying external;
      dcl working_dir character (168) external;

      dcl control_law_file_ptr pointer external;

      dcl i fixed;
      dcl code fixed binary (35);
      dcl control_law (1:num_state_combs) fixed based (control_l
aw_file_ptr);

      dcl delete entry options (variable);
      dcl hcs_$make_seg entry (char (*), char (*), char (*), fix
ed bin (5),
                                   ptr, fixed bin (35));

      call delete (job_name||".control_law", "-bf");
      call hcs_$make_seg (working_dir, job_name||".control_law",
 "", 01010b,
                               control_law_file_ptr, code);

      do i = 1 to num_state_combs;
        control_law (i) = 0;
      end;

    end open_control_law_file;


    find_loops_and_cont_law: procedure (cell_status, center_dist
,
                                                                   cel
l_status_index, center_cell_tolerance, found_all_loops);

      dcl cell_status (*) fixed;
      dcl center_dist (*) fixed;
      dcl cell_status_index fixed;
      dcl center_cell_tolerance fixed;

      dcl num_state_combs fixed external;
      dcl num_input_combs fixed external;

      dcl sat_edge (1:num_state_combs, 1:num_input_combs) bit(1)
 controlled
                    external;
      dcl next_state_file_ptr pointer external;
      dcl control_law_file_ptr pointer external;

      dcl i fixed;
      dcl state_code fixed;
      dcl input_code fixed;
      dcl next_state_code fixed;
      dcl root_code fixed;
      dcl root_status_code fixed;
      dcl min_cent_dist fixed;
      dcl best_cell_to_add fixed;
      dcl best_control_input fixed;
      dcl control_law_input (1:num_state_combs) fixed controlled
;

    dcl next_state_map (1:num_state_combs, 1:num_input_combs) fi
xed controlled external;
        dcl the_next_state_mapping (1:num_state_combs, 1:num_inp
ut_combs) fixed binary(18)
                                unsigned based (next_state_file_ptr);
      dcl control_law (1:num_state_combs) fixed based (control_l
aw_file_ptr);

      dcl found_all_loops bit(1);
      dcl found_root_code bit(1);
      dcl found_loop bit(1);
      dcl added_cell_to_tree bit(1);
      dcl true bit(1) initial ("1"b);
      dcl false bit(1) initial ("0"b);


      allocate control_law_input;

      found_all_loops = true;
      do root_status_code = 3 by 2 to cell_status_index
                                                  while (found_al
l_loops = true);
        found_root_code = false;
        do i = 1 to num_state_combs while (found_root_code = fal
se);
          if ( cell_status (i) = root_status_code ) then
            do;
              root_code = i;
              found_root_code = true;
            end;
        end;
        control_law_input = 0;
        found_loop = false;
        control_law_input (root_code) = 9999;
        do input_code = 1 to num_input_combs;
          next_state_code = next_state_map (root_code, input_cod
e);
          if ( next_state_code = root_code ) then
            do;
              found_loop = true;
              control_law_input (root_code) = input_code;
            end;
        end;
        added_cell_to_tree = true;
        do while ( found_loop = false & added_cell_to_tree = tru
e );
          min_cent_dist = center_cell_tolerance + 1;
          do state_code = 1 to num_state_combs;
            if ( cell_status (state_code) = root_status_code + 1
 &
                 control_law_input (state_code) = 0 &
                 center_dist (state_code) ^= 0 ) then
```

```
                do;
                  do input_code = 1 to num_inout_combs;
                    if ( sat_edge (state_code, input_code) = false
    ) then
                      do;
                        next_state_code = next_state_map (state_co
de, input_code);
                        if ( cell_status (next_state_code) = root_
status_code |
                        cell_status (next_state_code)= root_status
_code + 1 ) then
                          do;
                            if (control_law_inout (next_state_code
) ^= 0 ) then
                              do;
                                if (center_dist (next_state_code)
< min_cent_dist)
                                then
                                  do;
                                    min_cent_dist = center_dist (n
ext_state_code);
                                    best_cell_to_add = state_code;
                                    best_control_inout = input_cod
e;
                                  end;
                              end;
                          end;
                      end;
                  end;
                end;
            end;
            if ( min_cent_dist < center_cell_tolerance + 1 ) then
              do;
                added_cell_to_tree = true;
                control_law_input (best_cell_to_add) = best_contro
l_input;
                do input_code = 1 to num_input_combs;
                  next_state_code = next_state_map (root_code, inp
ut_code);
                  if ( next_state_code = best_cell_to_add ) then
                    do;
                      fourd_loop = true;
                      control_law_input (root_code) = input_code;
                    end;
                end;
              end;
            else
              do;
                added_cell_to_tree = false;
                found_loop = false;
              end;
          end;
          if ( found_loop = true ) then
            do;
              found_all_lcops = true;
              control_law (root_code) = control_law_input (root_co
de);
              state_code= next_state_map (root_code,control_law_in
out (root_code));
              do while ( state_code ^= root_code );
                control_law (state_code) = control_law_inout (stat
e_code);
                state_ccde = next_state_map (state_code, control_l
aw_input
  (state_code));
              end;
            end;
```

```
        else
          found_all_loops = false;
      end;
      if ( found_all_lccps = true ) then
        do;
          put edit ("Sucessfully built tolerant region control l
aw")(skip, x(4),

       a);
        end;
      else
        do;
          put edit ("Tolerant region control law cound not be bu
ilt")(skip, x(8),

       a);
        end;

      free control_law_input;

    end find_loops_anc_cont_law;


    build_optimal_contrcl_law: procedure (cell_status, cell_stat
us_index);

      dcl cell_status (*) fixed;
      dcl cell_status_index fixed;

      dcl num_state_combs fixed external;
      dcl num_input_combs fixed external;

      dcl sat_edge (1:num_state_combs, 1:num_input_combs) bit(1)
 controlled
                        external;

      dcl next_state_file_ptr pointer external;
      dcl control_law_file_ptr pointer external;

      dcl state_code fixed;
      dcl input_code fixed;
      dcl next_state_code fixed;
      dcl root_status_ccde fixed;

      dcl 1 min_path,
            2 cost float,
            2 sta_code fixed,
            2 inp_code fixed;

      dcl control_law (1:num_state_combs) fixed based (control_l
aw_file_ptr);
  dcl next_state_map (1:num_state_combs, 1:num_input_combs) fi
xed controlled external;
        dcl the_next_state_mapping (1:num_state_combs, 1:num_inp
ut_combs) fixed binary(18)
                                unsigned based (next_state_file_ptr);

      dcl cost float;
      dcl path_cost (1:num_state_combs) float controlled;

      dcl found_cell_to_add bit(1);
      dcl true bit(1) initial ("1"b);
      dcl false bit(1) initial ("0"b);

      dcl sysprint file output;


      allocate path_cost;
    put edit ("Building control law")(skip, x(4), a);
    path_cost = 0;
    do root_status_code = 3 by 2 to cell_status_index;
      found_cell_to_acd = true;
      do while ( fourc_cell_to_add= true );
        min_path.cost = 1e38;
        do state_code = 1 to num_state_combs;
          if ( cell_status (state_code) = root_status_code + 1
 &
                control_law (state_code) = 0 ) then
            do;
              do input_code = 1 to num_input_combs;
                if ( sat_edge (state_code, input_code) = false
 ) then
                  do;
                    next_state_code = next_state_map (state_co
de, input_code);
                    if ( cell_status (next_state_code) = root_
status_code |
                      cell_status (next_state_code)= root_stat
us_code + 1) then
                      do;
                        if ( control_law (next_state_code) ^=
0 &
                          next_state_code ^= state_code ) t
hen
                          do;
                            cost = compute_cost (state_code, i
nput_code);
                            cost = cost + path_cost (next_stat
e_code);
                            if ( cost < min_path.cost) then
                              do;
                                min_path.cost = cost;
                                min_path.sta_code = state_code
;
                                min_path.inp_code = input_code
;
                              end;
                          end;
                      end;
                  enc;
              end;
            end;
        end;
        if ( min_path.cost < 1e38 ) then
          do;
            found_cell_to_add = true;
            control_law (min_path.sta_code) = min_path.inp_cod
e;
            path_cost (min_path.sta_code) = min_path.cost;
          end;
        else
          found_cell_to_add = false;
      end;
    end;

    free path_cost;

    compute_cost: procedure (state_code, input_code) returns (
float);

      dcl state_code fixed;
      dcl input_code fixed;

      dcl n fixed external;
      dcl p fixed external;
```

```
        dcl cost_function_code fixed external;

        dcl state_cost_matrix (1:n) float controlled external;
        dcl input_cost_matrix (1:p) float controlled external;

        dcl i fixed;
        dcl dis_state (1:n) fixed controlled;
        dcl dis_input (1:p) fixed controlled;

        dcl cost float;
        dcl cont_state (1:n) float controlled;
        dcl cont_input (1:p) float controlled;

        dcl convert_$code_to_dis_state entry (fixed, (*) fixed);
        dcl convert_$code_to_dis_input entry (fixed, (*) fixed);
        dcl convert_$dis_state_to_cont_state entry ((*) fixed, (
*) float);
        dcl convert_$dis_input_to_cont_input entry ((*) fixed, (
*) float);
        dcl custom_cost_function entry ((*) float, (*) float) re
turns (float);


        goto case (cost_function_code);

     case (1): cost = 1;
               goto done;

     case (2): allocate dis_state, dis_input, cont_state, cont_
input;
               call convert_$code_to_dis_state ((state_code), d
is_state);
               call convert_$code_to_dis_input ((input_code), d
is_input);
               call convert_$dis_state_to_cont_state ((dis_stat
e), cont_state);
               call convert_$dis_input_to_cont_input ((dis_inpu
t), cont_input);
               cost = 0;
               do i = 1 to n;
                 cost = cost + ((cont_state (i) ** 2) * state_c
ost_matrix (i));
               end;
               do i = 1 to p;
                 cost = cost + ((cont_input (i) ** 2) * input_c
ost_matrix (i));
               end;
               free dis_state, dis_input, cont_state, cont_inpu
t;
               goto done;

     case (3): allocate dis_input, cont_input;
               call convert_$code_to_dis_input ((input_code), d
is_input);
               call convert_$dis_input_to_cont_input ((dis_inpu
t), cont_input);
               cost = 0;
               do i = 1 to p;
                 cost = cost + (input_cost_matrix (i) * abs (co
nt_input (i)));
               end;
               free dis_input, cont_input;
               goto done;

     case (4): allocate dis_state, dis_input, cont_state, cont_
input;
               call convert_$code_to_dis_state ((state_code), d
is_state);
```

```
                    call convert_$code_to_dis_input ((input_code), d
is_input);
                    call convert_$dis_state_to_cont_state ((dis_stat
e), cont_state);
                    call convert_$dis_input_to_cont_input ((dis_inpu
t), cont_input);
                    cost = custom_cost_function (cont_state, cont_in
put);
                    free dis_state, dis_input, cont_state, cont_inpu
t;
                    goto done;

    done:           return (cost);

    end compute_cost;

  end build_optimal_control_law;


  print_cell_status: procedure (cell_status);

    dcl cell_status (*) fixed;

    dcl n fixed external;
    dcl num_state_combs fixed external;
    dcl number_of_steps_s (1:n) fixed controlled external;

    dcl i fixed;
    dcl j fixed;
    dcl k fixed;
    dcl l fixed;

    dcl yn_answer_ok entry (character (3) varying);
    dcl sysprint file output;

    put edit ("Would you like the cell status array printed ?
 ")
                  (skip, a);
    get list (answer);
    call yn_answer_ok (answer);
    if ( answer = "y" | answer = "yes" ) then
      do;
        if ( n = 2 ) then
          do;
            k = 0;
            put skip (2);
            do i = 1 to number_of_steps_s (2);
              put skip;
              do j = 1 to number_of_steps_s (1);
                k = k + 1;
                put edit (cell_status (k))(f(5));
              end;
            end;
          end;
        else
          do;
            if ( n = 3 ) then
              do;
                l = 0;
                do i = 1 to number_of_steps_s (3);
                  put skip (2);
                  do j = 1 to number_of_steps_s (2);
                    put skip;
                    do k = 1 to number_of_steps_s (1);
                      l = l + 1;
                      put edit (cell_status (l))(f(5));
                    end;
                  end;
                end;
              end;
            else
              do;
                do i = 1 to num_state_combs;
                  put edit (i, cell_status (i))(skip, f(36), f
(6));
                end;
              end;
          end;
      end;

   end print_cell_status;


print_control_law:  procedure;

   dcl i fixed;
   dcl num_state_combs fixed external;
   dcl control_law_file_ptr pointer external;
   dcl answer character (3) varying;

   dcl yn_answer_ok entry (character(3) varying);
   dcl sysin file input;
   dcl sysout file output;
   dcl control_law (1:num_state_combs) fixed based (control_law
_file_ptr);
   put edit ("Would you like the control law printed? => ")(ski
p,a);
   get list (answer);
   call yn_answer_ok (answer);
   if (answer = "y" | answer = "yes") then
      call pr_cont_law;
  end print_control_law;

goto endit;
pr_cont_law: entry;
        put edit ("    Control Law:")(skip, a);
        put skip;
        do i = 1 to num_state_combs;
          put edit (i, control_law (i))(skip, x(4), f(4), f(6)
);
        end;
        put skip;

   return;


endit:  put skip;

end build_tol_reg_and_cont_law;
```

```
simulate_system: procedure;

  dcl n fixed external;
  dcl p fixed external;
  dcl p_real fixed external;
  dcl number_of_steps_s (1:n) fixed controlled external;
  dcl number_of_steps_i (1:p) fixed controlled external;
  dcl voltage_upper_bound_s (1:n) float controlled external;
  dcl voltage_lower_bound_s (1:n) float controlled external;
  dcl voltage_upper_bound_i (1:p) float controlled external;
  dcl voltage_lower_bound_i (1:p) float controlled external;
  dcl num_state_combs fixed external;
  dcl num_sim_data fixed external;

  dcl tau float external;
  dcl 1 simulation_data (1:num_sim_data) controlled external,
        2 time float,
        2 con_state (10) float,
        2 con_input (1:input_dim) float;

  dcl 1 flag external,
          2 cont_exists bit(1),
          2 discrete_exists bit(1),
          2 quantized_exists bit(1),
          2 control_law_valid bit(1),
          2 sim_valid bit(1),
          2 own_quant_file_exists bit(1);

  dcl true bit(1)   initial ("1"b);
  dcl false bit(1)   initial ("0"b);
  dcl own_cont_sys_exists bit(1);

  dcl uncontrollable_cell (1:num_state_combs) bit (1) controll
ed external;

  dcl control_law_file_ptr pointer external;

  dcl i fixed;
  dcl j fixed;
  dcl answer character (3) varying;
  dcl save_it character (3) varying;
  dcl choice character(1);
  dcl choice_value fixed;
  dcl range fixed;
  dcl max_num_steps fixed;
  dcl num_step fixed;
  dcl num_recurse_levels fixed;
  dcl region fixed;
  dcl matrix_dim fixed binary (35);
  dcl input_dim fixed external;
  dcl ind fixed binary (35);
  dcl ier fixed binary (35);
  dcl state_code fixed;
  dcl input_code fixed;
  dcl temp_state_code fixed;
  dcl dis_state (1:n) fixed controlled;
  dcl dis_input (1:p) fixed controlled;
  dcl temp_dis_state (1:n) fixed controlled;
  dcl control_law (1:num_state_combs) fixed based (control_law
_file_ptr);

  dcl time float binary;
  dcl time_init float;
  dcl time_final float;
  dcl step float;
  dcl step_end float binary;
  dcl tolerance float binary;
  dcl scale_factor float;
```

```
   dcl c (1:24) float binary;
   dcl cont_state (10) float binary;
   dcl state_temp(10) float binary;
   dcl input_to_use (1:input_dim) float controlled external;
   dcl cont_input (1:input_dim) float controlled external;
   dcl temp_cont_state (10) float binary;
   dcl w (1:matrix_dim, 1:9) float binary controlled;

   dcl print_data character (3) varying;

   dcl system_unstable bit (1);

   dcl convert_$cont_state_to_dis_state entry ((*) float, (*) f
ixed);
   dcl convert_$dis_state_to_code entry ((*) fixed, fixed);
   dcl convert_$code_to_dis_input entry (fixed, (*) fixed);
   dcl convert_$dis_input_to_cont_input entry ((*) fixed, (*) f
loat);
   dcl choose_your_plot entry;
   dcl yn_answer_ok entry (character(3) varying);
   dcl num_answer_ok entry (character (1), fixed, fixed);

   dcl own_sys_to_sim entry (fixed binary(35), float binary, fl
oat binary, (*) float binary, (*) float binary, float binary,
float binary);

   dcl imsl$dverk entry (fixed binary (35), entry, float binary
, (*) float
                              binary, float binary, float binary, fi
xed binary (35),
                              (*) float binary, fixed binary (35), (
*, *) float
                              binary, fixed binary (35));

   dcl sim_cont_file file;
   dcl sim_data_file file;
   dcl sysin file input;
   dcl sysprint file output;

   dcl mod builtin;

   if (flag.control_law_valid = false) then do;
     put edit ("A control law does not exist for this job")(ski
p,a);
     goto done;
   end;

   if (flag.sim_valid = false) then do;
      put edit("Would you like to simulate the system? => ")(sk
ip,a);
      get list (answer);
      call yn_answer_ok (answer);
      if (answer = "n" | answer = "no") then
         goto done;
   end;
   else do;
      put edit (" Would you like to : ")(skip,a);
      put skip;
      put edit ("1.  Modify the simulated data file")(skip,a);
      put edit ("2.  Plot your existing simulated data")(skip,a
);
      put edit ("3.  Quit")(skip,a);
      put skip;
      put edit ("Please choose one => ")(skip,a);
      get list (choice);
      range = 3;
      call num_answer_ck (choice, range, choice_value);
      if (choice_value = 1) then
```

```
          flag.sim_valid = false;
       if (choice_value = 2) then
          do;
             call choose_your_plot;
             goto done;
          end;
       if (choice_value = 3) then
          goto done;
   end;

   if (flag.quantized_exists = false) then do;
     put edit ("The parameters for the continuous system are ne
eded")(skip,a);
     put edit ("The system can not be simulated")(skip,a);
     goto done;
   end;


  if (flag.cont_exists = true) then do;
   put edit ("Would you like to simulate:  ")(skip,a);
   put edit ("     1.  The continuous system in the job file")(
skip,a);
   put edit ("     2.  A continuous system in another file")(sk
ip,a);
   put skip;
   put edit ("Please choose one =>  ")(skip,a);
   get list (choice);
   range = 2;
   call num_answer_ok (choice, range, choice_value);
  end;
  else choice_value = 2;

   if (choice_value = 2) then do;
     own_cont_sys_exists = true;
     put edit ("Enter the number of states => ")(skip,a);
     get list (matrix_dim);
     put edit ("Enter the number of inputs => ")(skip,a);
     get list (input_dim);
   end;
   if (choice_value = 1) then do;
     own_cont_sys_exists = false;
     matrix_dim = n;
     input_dim = p;
   end;

   if (tau = 0) then do;
     put edit ("Please enter tau  =>  ")(skip,a);
     get list (tau);
   end;

  allocate dis_state, dis_input, temp_dis_state, cont_input, in
put_to_use;

   put edit ("Enter number of steps per time constant  => ")(sk
ip, a);
   get list (max_num_steps);
   put edit ("Enter the number of recursion levels => ")(skip,
a);
   get list (num_recurse_levels);
   if (num_recurse_levels ^= 0) then do;
     put edit ("Enter the scaling factor => ")(skip, a);
     get list (scale_factor);
   end;
   else
     scale_factor = 1e-20;
   put edit ("Enter initial state")(skip, a);
   put skip;
   do i = 1 to matrix_dim;
```

```
         put edit ("initial state ",i," => ")(x(4), a, f(3), a);
         get list (cont_state (i));
         state_temp (i) = cont_state (i);
      end;
      put edit ("Enter initial time => ")(skip, a);
      get list (time_init);
      put edit ("Enter final time => ")(skip, a);
      get list (time_final);
      put edit ("Would you like the simulation printed while runni
  ng => ")(skip, a);
      get list (print_data);
      call yn_answer_ok (print_data);
      put skip;
      put edit ("   Simulating system")(skip, x(2), a);
      if ( print_data = "y" | print_data = "yes" ) then
         put edit ("time","state", "input")(skip(2), x(3), a,x(12),
   a, x(10), a);
      put skip;
      time = time_init;
      step = tau / max_num_steps;
      num_sim_data = ceil ((time_final - time_init) / step);
      if ((num_sim_data + 1) * step <= time_final) then
         num_sim_data = num_sim_data + 1;

      allocate simulation_data;

      num_step = 0;
      ind = 1;
      tolerance = .0001;
      system_unstable = false;
      do i = 1 to num_sim_data while ( system_unstable = false );
         do j = 1 to n;
            if ( cont_state (j) < voltage_lower_bound_s (j) |
                 cont_state (j) > voltage_upper_bound_s (j) ) then
               system_unstable = true;
         end;
         if ( system_unstable = false ) then
            do;
               call convert_$cont_state_to_dis_state ((cont_state), d
  is_state);
               call convert_$dis_state_to_code ((dis_state), state_co
  de);
               if ( uncontrollable_cell (state_code) = false ) then
                  do;
                     if (num_step = 0) then
                        do;
                           call find_region (cont_state, num_recurse_leve
  ls, scale_factor,

               region);
                           temp_cont_state = cont_state / (scale_factor *
  * region);
                           call convert_$cont_state_to_dis_state (temp_co
  nt_state,

  temp_dis_state);
                           call convert_$dis_state_to_code (temp_dis_stat
  e,
                                                                              t
  emp_state_code);
                           input_code = control_law (temp_state_code);
                           call convert_$code_to_dis_input ((input_code),
   dis_input);
                           call convert_$dis_input_to_cont_input ((dis_in
  put),cont_input);

                           cont_input = cont_input * (scale_factor ** reg
  ion);
```

```
                   end;
                 simulation_data (i).time = time;
                 simulaticn_data (i).con_state (*) = cont_state (*)
;
                 simulaticn_data (i).con_input (*) = cont_input (*)
;


                 if ( print_data = "y" | print_data = "yes" ) then
                   do;
                     put edit (time)(skip, f(8,3));
                     put edit ("  ")(a);
                     do j = 1 to matrix_dim;
                       put edit (cont_state (j))(f(14,3));
                     end;
                     put edit (" ")(a);
                     do j = 1 to input_dim;
                       put edit (cont_input (j))(f(14,3));
                     end;
                   end;
                 step_end = time + step;
                 if (own_cont_sys_exists = true) then do;
                   input_to_use = cont_input;
                   call own_sys_to_sim (matrix_dim, time, step_end,
 state_temp, cont_state, time_init, time_final);
                 end;
                 else do;
                   allocate w;
                   call imsl$dverk (matrix_dim, cont_system, time,
state_temp, step_end,
                                  tolerance, ind, c, matrix_dim, w,
 ier);
                   cont_state = state_temp;
                   free w;
                   if (inc < 0 | ier > 0) then do;
                     put edit ("ERROR using IMSL")(skip,a);
                     put edit ("ind =  ")(skip,a);
                     put list (ind);
                     put edit ("ier  = ")(skip,a);
                     put list (ier);
                   end;
                 end;
               end;
             else
                 system_unstable = true;
          end;
          num_step = num_step + 1;
          if (num_step = max_num_steps) then
            num_step = 0;
       end;
       if (system_unstable = true ) then
         do;
           put edit ("The system has gone unstable")(skip(2), a);
               call build_sim_data_file;
               call choose_your_plot;
         end;
       else
         do;
           call build_sim_data_file;
           call choose_your_plot;
         end;

       free dis_state, dis_input, temp_dis_state, cont_input, simul
ation_data, input_to_use;

done:   put skip (2);

       find_region : procedure (cont_state, num_recurse_levels, sca
le_factor,
          region);

     dcl cont_state (*) float;
     dcl num_recurse_levels fixed;
     dcl scale_factor float;
     dcl region fixed;

     dcl n fixed external;

     dcl voltage_upper_bound_s (1:n) float controlled external;
     dcl voltage_lower_bound_s (1:n) float controlled external;

     dcl i fixed;
     dcl j fixed;

   region = num_recurse_levels;
   if (num_recurse_levels ^= 0) then do;
     do i = 1 to n;
       if ( cont_state (i) >= 0 ) then
         do;
           do j = 1 to num_recurse_levels while ( cont_state (i
) <
                                 ( voltage_upper_bound_s (i) * (scale_
factor ** j )));
           end;
         end;
       else
         do;
           do j = 1 to num_recurse_levels while ( cont_state (i
) >
                                 ( voltage_lower_bound_s (i) * (scale_
factor ** j )));
           end;
         end;
       if ( j-1 < region ) then
         region = j-1;
     end;
   end;

   end find_region;

   cont_system: procedure (matrix_dim, time, state_temp, state_
temp_prime);

     dcl matrix_dim fixed binary (35);
     dcl time float binary;
     dcl state_temp (10) float binary;
     dcl state_temp_prime (10) float binary;

     dcl n fixed external;
     dcl p fixed external;

     dcl cont_input (1:p) float controlled external;
     dcl a_matrix (1:n, 1:n) float controlled external;
     dcl b_matrix (1:n, 1:p) float controlled external;

     dcl i fixed;
     dcl j fixed;


     do i = 1 to n;
       state_temp_prime (i) = 0;
       do j = 1 to n;
         state_temp_prime (i) = state_temp_prime (i) + (a_matri
x (i, j) *
                                        state_temp (j));
          end;
        end;
        do i = 1 to n;
          do j = 1 to p;
             state_temp_prime (i) = state_temp_prime (i) + (b_matri
x (i, j) *
                                                 cont_input (j));
          end;
        end;

      end cont_system;

build_sim_data_file:  procedure;

      dcl n fixed external;
      dcl p fixed external;
      dcl num_sim_data fixed external;


      dcl 1 simulation_data (1:num_sim_data) controlled external,
               2 time float,
               2 con_state (1:n) float,
               2 con_input (1:p) float;

      dcl true bit(1)   initial ("1"b);
      dcl false bit(1)   initial ("0"b);
      dcl 1 flag external,
               2 cont_exists bit(1),
               2 discrete_exists bit(1),
               2 quantized_exists bit(1),
               2 control_law_valid bit(1),
               2 sim_valid bit(1),
               2 own_quant_file_exists bit(1);

      dcl job_name character (50) varying external;

      dcl width_sim_mat fixed;

      dcl sim_data_matrix (1:num_sim_data, 1:width_sim_mat) float
controlled external;

      dcl answer character (3) varying;
      dcl i fixed;
      dcl j fixed;
      dcl k fixed;


      dcl yn_answer_ok entry (character(3) varying);
      dcl sim_data_file file;
      dcl sysin file input;
      dcl sysprint file output;

      width_sim_mat = n + p + 1;
      allocate sim_data_matrix;

           sim_data_matrix (*,1) = simulation_data (*).time;
           do j = 2 to (n+1);
              sim_data_matrix (*,j) = simulation_data (*).con_state
(j-1);
           end;
           do k = (n+2) to (n+p+1);
              sim_data_matrix (*,k) = simulation_data (*).con_input
(k-n-1);
           end;
      put edit ("Would you like to save the simulated data in a fi
le? => ")(skip,a);
      get list (answer);
      call yn_answer_ok (answer);
```

```
   if (answer= "y" | answer = "yes") then do;

   open file (sim_data_file) title ("vfile_ "||job_name||
                                     "_ts.plot") stream output;


   do i = 1 to num_sim_data;
      do j = 1 to width_sim_mat;
         put file (sim_data_file) list (sim_data_matrix (i,j));
      end;
   end;

   free sim_data_matrix;
   flag.sim_valid = true;
   end;

   close file(sim_data_file);

end build_sim_data_file;

end simulate_system;
```

```
choose_your_plot: procedure;

   dcl n fixed external;
   dcl p fixed external;
   dcl num_sim_data fixed external;

   dcl job_name character (50) varying external;

   dcl true bit(1)    initial ("1"b);
   dcl false bit(1)    initial ("0"b);
   dcl 1 flag external,
         2 cont_exists bit(1),
         2 discrete_exists bit(1),
         2 quantized_exists bit(1),
         2 control_law_valid bit(1),
         2 sim_valic bit(1),
         2 own_quant_file_exists bit(1);

   dcl sim_data_matrix (1:num_sim_data, 1:width_sim_mat) float
controlled external;

   dcl range fixed;
   dcl y_axis_choice fixed;
   dcl y_axis_c character(1);
   dcl x_axis_choice fixed;
   dcl x_axis_c character(1);
   dcl width_sim_mat fixed;
   dcl number_of_plots fixed;
   dcl answer character(3) varying;

   dcl plot_x fixed;
   dcl plot_y fixed;

   dcl x_array (1:number_of_plots, 1:num_sim_data) float contro
lled;
   dcl y_array (1:number_of_plots, 1:num_sim_data) float contro
lled;


   dcl i fixed;
   dcl j fixed;
   dcl l fixed;

   dcl plot_data character (3) varying;

   dcl sim_data_file file;
   dcl sysin file input;
   dcl sysprint file cutput;
   dcl num_answer_ok entry (character(1), fixed, fixed);
   dcl yn_answer_ok entry (character(3) varying);

   width_sim_mat = n + p + 1;

    put edit ("Would ycu like to plot the simulated data? => ")
(skip, a);
    get list (plot_data);
    call yn_answer_ok (plot_data);
    if (plot_data = "yes" | plot_data = "y") then do;
       if (flag.sim_valid = true) then do;
          allocate sim_data_matrix;
          open file (sim_data_file) title ("vfile_ "||job_name||
"_ts.plot") stream input;
          do i = 1 to num_sim_data;
             do j = 1 to width_sim_mat;
                get file (sim_data_file) list (sim_data_matrix (
i,j));
             end;
          end;
```

```
        end;
      end;


      do while ( plot_data = "yes" | plot_data = "y" );
       if ( plot_data = "yes" | plot_data = "y" ) then
        do;
          put edit ("Would you like multiple plots on one graph? "
)(skip, a);
          get list (answer);
          call yn_answer_ok (answer);
          if (answer = "y" | answer = "yes") then
            do;
              put edit ("How many plots would you like to put on t
he graph? => ")(skip,a);
              get list (number_of_plots);
             end;
          else
            number_of_plots = 1;

         allocate x_array;
         allocate y_array;


          do l = 1 to number_of_plots;
              if (number_of_plots > 1) then
                put edit ("PLOT ", l, ":")(skip, a, x(1), f(1), x(
1), a);
             put edit("What would you like to plot on the y axis?
")(skip,a);
             put edit("1.  A state")(skip,a);
             put edit("2.  An input")(skip,a);
             put edit("3.  Time")(skip,a);
             put skip;
             put edit ("Please choose one => ")(skip,a);
             get list (y_axis_c);
             range = 3;
             call num_answer_ok (y_axis_c, range, y_axis_choice);

             if (y_axis_choice = 1) then
                do;
                   if (n = 1) then
                      plot_y = 1;
                   else
                      do;

                         put edit("Which state do you wish to plot
 on the y axis? ")(skip,a);
                         get list (plot_y);
                      end;
                   plot_y = plot_y + 1;
                end;

             if (y_axis_choice = 2) then
                do;
                   if (p = 1) then
                      plot_y = 1;
                   else
                      do;
                         put edit("Which input do you wish to plot
 on the y axis? ")(skip, a);
                         get list (plot_y);
                      end;
                   plot_y = plot_y + n + 1;
                end;

               if (y_axis_choice = 3) then
```

```
                 plot_y = 1;

          put edit ("What would you like to plot on the x axis?
")(skip,a);
          put edit("1.   A state")(skip,a);
          put edit("2.   An input")(skip,a);
          put edit("3.   Time")(skip,a);
          put skip;
          put edit ("Please choose one => ")(skip,a);
          get list (x_axis_c);
          call num_answer_ok (x_axis_c, range, x_axis_choice);

          if (x_axis_choice = 1 ) then
             do;
               if(n = 1) then
                  plot_x = 1;
               else
                  do;
                      put edit("Which state do you wish to plot
 on the x_axis => ")(skip,a);
                      get list (plot_x);
                  end;
               plot_x = plot_x + 1;
             end;

          if (x_axis_choice = 2) then
             do;
                if (p = 1) then
                   plot_x = 1;
                else
                   do;
                      put edit("Which input do you wish to plot
 on the x axis?")(skip,a);
                      get list (plot_x);
                   end;
                plot_x = plot_x + n + 1;
             end;

            if (x_axis_choice = 3) then
               plot_x = 1;


          x_array(l,*) = sim_data_matrix(*, plot_x);
          y_array (l,*) = sim_data_matrix(*, plot_y);

       end; /* do loop */

          call plot_the_sim (x_array, y_array, number_of_plots)
;
          free x_array;
          free y_array;
          put skip (3);

          put edit ("Would you like to plot the simulated data?
 => ")(skip, a);
          get list (plot_data);
          call yn_answer_ok (plot_data);

     end;

   end;  /* while */
   close file (sim_data_file);



plot_the_sim: procedure (x_array, y_array, number_of_plots);
```

```
   dcl x_array (*,*) float parameter;
   dcl y_array (*,*) float parameter;
   dcl number_of_plots fixed parameter;

   dcl num_sim_data fixed external;

   dcl x(1:num_sim_data) float controlled external;
   dcl y(1:num_sim_data) float controlled external;

   dcl vec_sw fixed bin;
   dcl symbol (1:number_of_plots) character(1) controlled;
   dcl symbol_mark character(1);
   dcl l_char character(1);

   dcl scale_auto bit(1);
   dcl true bit(1) initial ("1"b);
   dcl false bit(1) initial ("0"b);
   dcl xmin float bin;
   dcl xmax float bin;
   dcl ymin float bin;
   dcl ymax float bin;

   dcl l fixed;
   dcl answer character (3) varying;
   dcl graph_title character (25);
   dcl xlabel character (25);
   dcl ylabel character (25);
   dcl graph_type fixec bin;
   dcl base float bin;
   dcl grid_sw_char character (1);
   dcl grid_sw fixed bin;
   dcl eq_scale_sw fixed bin;

   dcl sysin file input;
   dcl sysprint file output;

   dcl num_answer_ok entry (character(1), fixed, fixed);
   dcl yn_answer_ok entry (character(3) varying);
   dcl plot_ entry ((*) float bin, (*) float bin, fixed bin, fi
xed bin, char(1));
   dcl plot_$scale entry (float bin, float bin, float bin, floa
t bin);
   dcl plot_$setup entry (char(*), char(*), char(*), fixed bin,
 float bin, fixed bin, fixed bin);


   graph_title = "  ";
   xlabel = "  ";
   ylabel = "  ";
   scale_auto = true;
   graph_type = 1;
   base = 0;
   grid_sw = 0;
   eq_scale_sw = 0;

   allocate symbol;
   symbol = "+";

   put edit ("Would ycu like a symbol to represent each data po
int? => ")(skip,a);
   get list (answer);
   call yn_answer_ok (answer);
   if (answer = "y" | answer = "yes") then
       do;
           do l = 1 to number_of_plots;
               if (number_of_plots < 2 ) then do;
                   put edit ("Enter the desired symbol  => ")(skip,
a);
```

```
                  get list (symbol(l));
               end;
               else do;
                  put edit ("Enter the desired symbol for Plot ",l
,"  => ")(skip, a, f(1), a);
                  get list (symbol(l));
               end;
            end;
            put edit ("Would you like the symbols to be connected
by vectors? => ")(skip,a);
            get list (answer);
            call yn_answer_ok (answer);
            if (answer = "y" | answer = "yes" ) then
               vec_sw = 2;
            else
               vec_sw = 3;
         end;
      else
         vec_sw = 1;


            put edit ("The graph will have tick marks, be automati
cally scaled," )(skip,a);
            put edit ("    and have no labels ")(skip,a);
            put edit ("Would you like to change any of these defau
lt options? => ")(skip,a);
            get list (answer);
            call yn_answer_ok (answer);
            if (answer = "y" | answer = "yes") then
               do;
                  put edit ("Would you like: ")(skip,a);
                  put skip;
                  put edit ("1.  Tick marks and values")(skip,a);
                  put edit ("2.  Dotted grid and values")(skip,a);
                  put edit ("3.  Solid grid and values")(skip,a);
                  put skip;
                  put edit ("Please choose one => ")(skip,a);
                  get list (grid_sw_char);
                  range = 3;
                  call num_answer_ok (grid_sw_char, range, grid_sw
);
                  grid_sw = grid_sw - 1;


                  put edit ("Would you like to enter a title and a
xis labels for your plot? => ")(skip, a);
                  get list (answer);
                  call yn_answer_ok (answer);
                  if (answer = "y" | answer = "yes") then
                     do;
                        put edit ("Enter the desired title for you
r plot")(skip,a);
                        put skip;
                        get list(graph_title);
                        put edit ("Enter the label for the x-axis"
)(skip,a);
                        put skip;
                        get list(xlabel);
                        put edit ("Enter the label for the y-axis"
)(skip,a);
                        put skip;
                        get list(ylabel);
                     end;

                  put edit ("Would you like to set the scale of th
e graph? => ")(skip,a);
                  get list (answer);
```

```
                    call yn_answer_ok (answer);
                    if (answer = "y" | answer = "yes") then
                       do;
                          scale_auto = false;
                          put edit ("Enter the lower bound of the x-
axis => ")(skip,a);
                          get list (xmin);
                          put edit ("Enter the upper bound of the x-
axis => ")(skip,a);
                          get list (xmax);
                          put edit ("Enter the lower bound of the y-
axis => ")(skip,a);
                          get list (ymin);
                          put edit ("Enter the upper bound of the y-
axis => ")(skip,a);
                          get list (ymax);
                       end;
              end;

           call plot_$setup (graph_title, xlabel, ylabel, graph_t
ype, base, grid_sw, eq_scale_sw);

           if (scale_auto = false) then
              call plot_$scale (xmin, xmax, ymin, ymax);

           allocate x;
           allocate y;



           do l = 1 to number_of_plots;
              x(*) = x_array(l,*);
              y(*) = y_array(l,*);
              symbol_mark = symbol(l);
              call plot_ (x, y, num_sim_data,vec_sw, symbol_mark)
;
           end;

           free x, y, symbol;

end plot_the_sim;

end choose_your_plot;
```

```
own_sys_to_sim:  procedure (matrix_dim, time, step_end, state_
temp,
                                 cont_state, time_init, time_final)
;

   dcl matrix_dim fixed binary (35);
   dcl time float binary;
   dcl step_end float binary;
   dcl state_temp (10) float binary;
   dcl cont_state (10) float binary;
   dcl time_init float binary;
   dcl time_final float binary;

   dcl ind fixed binary (35);
   dcl w(1:matrix_dim, 1:9) float binary controlled;
   dcl tolerance float binary;
   dcl c (1:24) float binary;
   dcl ier fixed binary (35);

   dcl input_dim fixed external;

   dcl pi float;
   dcl amplitude float;
   dcl freq float;
   dcl disturb float;
   dcl j fixed;

   dcl disturb_data file;
   dcl sysprint file output;
   dcl imsl$dverk entry (fixed binary (35), entry, float binary
, (*) float
                           binary, float binary, float binary, fi
xed binary (35),
                           (*) float binary, fixed binary (35), (
*,*)float
                           binary, fixed binary (35));

   pi = 3.1415927;
   amplitude  =  0.1;
   freq  =  20;
   tolerance = 0.001;
   ind = 1;
   if (time = time_init) then do;
     disturb  =  amplitude * sin(2*pi*freq*time);
     put edit ("AMPLITUDE = ")(skip,a);
     put list (amplituce);
     put edit ("FREQ = ")(skip,a);
     put list (freq);
   end;

   allocate w;

   call imsl$dverk (matrix_dim, own_cont_system1, time, state_t
emp, step_end,
                         tolerance, ind, c, matrix_dim, w, ier);

     disturb  =  amplitude * sin(2*pi*freq*time);

   cont_state(1)  =  state_temp (1) - disturb;
   free w;

   if (ind < 0 | ier > 0) then do;
     put edit ("time = ")(skip,a);
     put list (time);
     put edit ("ERROR!! using IMSL ")(skip,a);
     put edit ("ind = ")(skip,a);
```

```
      put list (ind);
      put edit ("ier = ")(skip,a);
      put list (ier);
    end;


    own_cont_system1: procedure (matrix_dim, time, state_temp, s
tate_temp_prime);
     /*  third order   */

      dcl matrix_dim fixed binary (35);
      dcl time float binary;
      dcl state_temp (10) float binary;
      dcl state_temp_prime (10) float binary;

    dcl input_to_use (1:input_dim) float controlled external;
      dcl cont_input (1:input_dim) float controlled external;

      dcl input_dim fixed external;
      dcl own_n fixed;
      dcl own_p fixed;

      dcl own_a_matrix (1:own_n, 1:own_n) float controlled;
      dcl own_b_matrix (1:own_n, 1:own_p) float controlled;

      dcl i fixed;
      dcl j fixed;

      own_n = 3;
      own_p = 1;
      allocate own_a_matrix, own_b_matrix;

      own_a_matrix (1,1) = 0;
      own_a_matrix (1,2) = 1;
      own_a_matrix (1,3) = 0;
      own_a_matrix (2,1) = 0;
      own_a_matrix (2,2) = 0;
      own_a_matrix (2,3) = 1;
      own_a_matrix (3,1) = -249778.14;
      own_a_matrix (3,2) = -140645.6;
      own_a_matrix (3,3) = -181.98;

      own_b_matrix (1,1) = 0;
      own_b_matrix (2,1) = 0;
      own_b_matrix (3,1) = 55493.2;


      do i = 1 to own_n;
        state_temp_prime (i) = 0;
        do j = 1 to own_n;
          state_temp_prime (i) = state_temp_prime (i) + (own_a_m
atrix (i, j) *
                                 state_temp (j));
        end;
      end;
      do i = 1 to own_n;
        do j = 1 to own_p;
          state_temp_prime (i) = state_temp_prime (i) + (own_b_m
atrix (i, j) *
                                 input_to_use (j));
        end;
      end;
      free own_a_matrix, own_b_matrix;
    end own_cont_system1;

    own_cont_system2: procedure (matrix_dim, time, state_temp, s
tate_temp_prime);
     /*  fric system   */
```

```
      dcl matrix_dim fixed binary (35);
      dcl time float binary;
      dcl state_temp (10) float binary;
      dcl state_temp_prime (10) float binary;

      dcl n fixed external;
      dcl p fixed external;

   dcl input_to_use (1:p) float controlled external;
      dcl cont_input (1:p) float controlled external;
      dcl own_a_matrix (1:n, 1:n) float controlled;
      dcl own_b_matrix (1:n, 1:p) float controlled;

      dcl fric float;
      dcl i fixed;
      dcl j fixed;


      allocate own_a_matrix, own_b_matrix;
      own_a_matrix = 0;
      own_b_matrix = .395;
      if (state_temp(1) > 0.01) then
        fric = 0.0917;
      else do;
        if (state_temp(1) < -0.01) then
          fric = -0.0917;
        else do;
          own_a_matrix = -9.17;
          fric = 0;
        end;
      end;

      do i = 1 to n;
        state_temp_prime (i) = 0;
        do j = 1 to n;
          state_temp_prime (i) = state_temp_prime (i) + (own_a_m
atrix (i, j) *
                                  state_temp (j));
        end;
      end;

      do i = 1 to n;
        do j = 1 to p;
          state_temp_prime (i) = state_temp_prime (i) + (own_b_m
atrix (i, j) *
                                  input_to_use (j)) - fric;
        end;
      end;
      free own_a_matrix, own_b_matrix;
   end own_cont_system2;

   own_cont_system3: procedure (matrix_dim, time, state_temp, s
tate_temp_prime);

      dcl matrix_dim fixed binary (35);
      dcl time float binary;
      dcl state_temp (10) float binary;
      dcl state_temp_prime (10) float binary;

      dcl n fixed external;
      dcl p fixed external;

   dcl input_to_use (1:p) float controlled external;
      dcl cont_input (1:p) float controlled external;
      dcl a_matrix (1:n, 1:n) float controlled external;
      dcl b_matrix (1:n, 1:p) float controlled external;
      dcl i fixed;
      dcl j fixed;

      do i = 1 to n;
        state_temp_prime (i) = 0;
        do j = 1 to n;
          state_temp_prime (i) = state_temp_prime (i) + (a_matri
x (i, j) *
                                                        state_temp (j));
        end;
      end;
      do i = 1 to n;
        do j = 1 to p;
          state_temp_prime (i) = state_temp_prime (i) + (b_matri
x (i, j) *
                                                        input_to_use (j));
        end;
      end;

    end own_cont_system3;

  end own_sys_to_sim;
```

```
convert_: procedure;

  dcl n fixed external;
  dcl p fixed external;
  dcl offset_s (1:n) fixed controlled external;
  dcl offset_i (1:p) fixed controlled external;

  dcl voltage_lower_bound_s (1:n) float controlled external;
  dcl voltage_lower_bound_i (1:p) float controlled external;
  dcl quantum_step_size_s (1:n) float controlled external;
  dcl quantum_step_size_i (1:p) float controlled external;

  dcl i fixed;
  dcl state_code fixed;
  dcl input_code fixed;
  dcl dis_state (*) fixec;
  dcl dis_input (*) fixed;

  dcl cont_state (*) float;
  dcl cont_input (*) float;


  cont_state_to_dis_state: entry (cont_state, dis_state);

    do i = 1 to n;
    dis_state(i) = floor ((cont_state(i) - voltage_lower_bound
_s(i))
                                                        / quantu
m_step_size_s(i));
    end;


  return;


  dis_state_to_code: entry (dis_state, state_code);

    state_code = 1;
    do i = 1 to n;
      state_code = state_code + (dis_state (i) * offset_s (i))
;
    end;

  return;


  code_to_dis_state: entry (state_code, dis_state);

    state_code = state_code - 1;
    do i = n by -1 to 1;
      dis_state (i) = floor (state_code / offset_s (i));
      state_code = mod (state_code, offset_s (i));
    end;

  return;


  dis_state_to_cont_state: entry (dis_state, cont_state);

    do i = 1 to n;
    cont_state(i) = ((dis_state(i) + 0.5) * quantum_step_size_
s(i))
                                                     + voltage_l
ower_bound_s(i);
    end;

  return;
  cont_input_to_dis_input: entry (cont_input, dis_input);

    do i = 1 to p;
    dis_input(i) = floor ((cont_input(i) - voltage_lower_bound
_i(i))
                                                        / quantu
m_step_size_i(i));
    end;

  return;


  dis_input_to_code: entry (dis_input, input_code);

    input_code = 1;
    do i = 1 to p;
      input_code = input_code + (dis_input (i) * offset_i (i))
;
    end;

  return;


  code_to_dis_input: entry (input_code, dis_input);

    input_code = input_code - 1;
    do i = p by -1 to 1;
      dis_input (i) = floor (input_code / offset_i (i));
      input_code = mod (input_code, offset_i (i));
    end;

  return;


  dis_input_to_cont_input: entry (dis_input, cont_input);

    do i = 1 to p;
    cont_input(i) = ((dis_input(i) + 0.5) * quantum_step_size_
i(i))
                                                     + voltage_l
ower_bound_i(i);
    end;

  return;

end convert_;
```

```
num_answer_ok:  procedure (c, range, choice);

   dcl c character (1);
   dcl range fixed;
   dcl choice fixed;

   dcl i fixed;
   dcl good_answer_flag bit(1);
   dcl true bit(1)   initial ("1"b);
   dcl false bit(1)   initial ("0"b);

   dcl sysin file input;
   dcl sysprint file output;


   good_answer_flag = false;
   do while (good_answer_flag = false);
      if (c >= "0" & c <= "9") then
         do;
            choice = c;
            do i = 1 by 1 to range;
               if (choice = i) then
                  good_answer_flag = true;
            end;
         end;
      if (good_answer_flag = false) then
         do;
            put edit ("Incorrect Response","Try Again => ")
                                                   (skip, a, skip, a);
            get list (c);
         end;
      else
         choice = c;
   end; /*while*/

end num_answer_ok;
```

```
yn_answer_ok: procedure (answer);

  dcl answer character (3) varying;

  dcl good_answer_flag bit(1);
  dcl true bit(1)  initial  ("1"b);
  dcl false bit(1)  initial ("0"b);

  dcl sysin file input;
  dcl sysprint file output;

  if (answer = "y" | answer = "yes" | answer = "no" | answer =
 "n")
     then good_answer_flag = true;
  else
     good_answer_flag = false;

  do while (good_answer_flag = false);
     put edit ("Incorrect Response", "Try Again => ")
                                       (skip, a, skip, a);
     get list (answer);
     if (answer = "y" | answer = "yes" | answer = "no" |
                                        answer = "n") then
        good_answer_flag = true;
     else
        good_answer_flag = false;
  end;   /*while*/

end yn_answer_ok;
```

DISTRIBUTION LIST

	Copies
Commander Defense Technical Information Center Bldg. 5, Cameron Station ATTN: DDAC Alexandria, VA 22314	12
Manager Defense Logistics Studies Information Exchange ATTN: AMXMC-D Fort Lee, VA 23801-6044	2
Commander U.S. Army Tank-Automotive Command ATTN: AMSTA-TSL Warren, MI 48397-5000	2
Commander U.S. Army Tank-Automotive Command ATTN: AMSTA-CV (COL Burke) Warren, MI 48397-5000	1
Chief System Simulation and Technology Division ATTN: AMSTA-RY Warren, MI 48397-5000	23